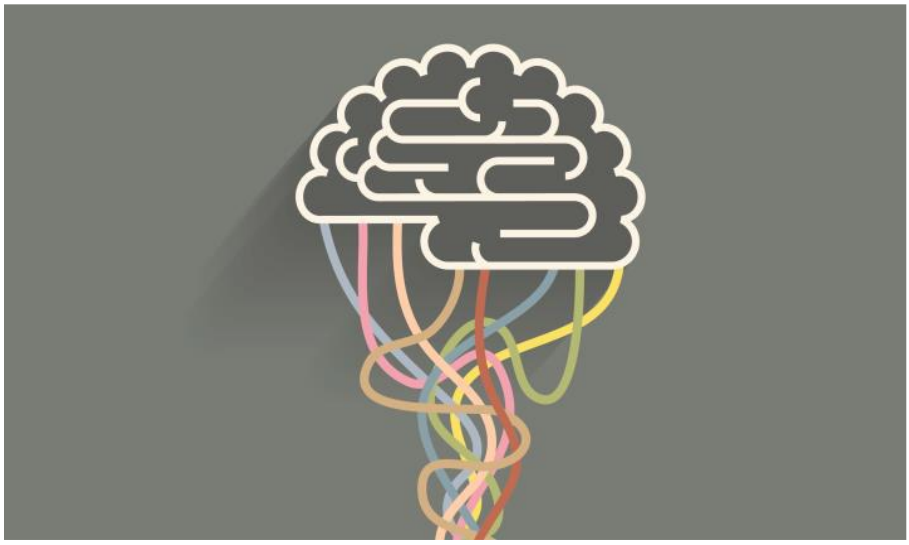
Numenta-Research-Papers

Where Neuroscience Meets Machine Intelligence



Numenta's deep experience in theoretical neuroscience research has led to tremendous discoveries about how the brain works. We are applying the principles of real intelligence to the world of artificial intelligence to create machines that can understand the world and add great value to humanity.

Contents

Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark

Alexander Lavin
Numenta, Inc. Redwood City, CA
alavin@numenta.com

Subutai Ahmad
Numenta, Inc. Redwood City, CA
sahmad@numenta.com

October 9, 2015

We appreciate and encourage any comments on the contents of this paper. The GitHub URL for NAB is https://github.com/numenta/NAB. The repository contains all the code, data, and detailed results reported in this paper. You can submit issues or pull requests there, or contact us at the above email addresses.

Note: This is a draft preprint of a paper to be published in ICMLA 2015, December 9-11, Miami, Florida. The final paper may be slightly different from this version. Please use the following citation for this paper:

A. Lavin and S. Ahmad, "Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark," in *14th International Conference on Machine Learning and Applications (IEEE ICMLA'15)*, 2015.

Evaluating Real-time Anomaly Detection Algorithms – the Numenta Anomaly Benchmark

Alexander Lavin
Numenta, Inc. Redwood City, CA
alavin@numenta.com

Subutai Ahmad
Numenta, Inc. Redwood City, CA
sahmad@numenta.com

Abstract

Much of the world's data is streaming, time-series data, where anomalies give significant information in critical situations; examples abound in domains such as finance, IT, security, medical, and energy. Yet detecting anomalies in streaming data is a difficult task, requiring detectors to process data in real-time, not batches, and learn while simultaneously making predictions. There are no benchmarks to adequately test and score the efficacy of real-time anomaly detectors. Here we propose the Numenta Anomaly Benchmark (NAB), which attempts to provide a controlled and repeatable environment of open-source tools to test and measure anomaly detection algorithms on streaming data. The perfect detector would detect all anomalies as soon as possible, trigger no false alarms, work with real-world time-series data across a variety of domains, and automatically adapt to changing statistics. Rewarding these characteristics is formalized in NAB, using a scoring algorithm designed for streaming data. NAB evaluates detectors on a benchmark dataset with labeled, real-world time-series data. We present these components, and give results and analyses for several open source, commercially-used algorithms. The goal for NAB is to provide a standard, open source framework with which the research community can compare and evaluate different algorithms for detecting anomalies in streaming data.

Keywords—*anomaly detection; time-series data; benchmarks; streaming data*

I. INTRODUCTION

With the rapid rise in real-time data sources the detection of anomalies in streaming data is becoming increasingly important. Use cases such as preventative maintenance, fraud prevention, fault detection, and monitoring can be found throughout numerous industries such as finance, IT, security, medical, energy, e-commerce, and social media. Anomaly detection is notoriously difficult to benchmark and compare [1, 2]. In addition, real-time applications impose their own unique constraints and challenges that must be considered. The goal of this paper is to introduce the Numenta Anomaly Benchmark (NAB), a rigorous new benchmark and source code for evaluating real-time anomaly detection algorithms.

Anomaly detection in real-world streaming applications is challenging. The detector must process data and output a decision in real-time, rather than making many passes through batches of files. In most scenarios the number of sensor streams is large and there is little opportunity for human, let alone expert, intervention. As such, operating in an unsupervised, automated fashion (e.g. without manual parameter tweaking) is often a necessity. As part of this automation, the detectors should continue to learn and adapt to changing statistics while simultaneously making predictions. The real goal is often prevention, rather than detection, so it is desirable to detect anomalies as early as possible, giving actionable information ideally well before a catastophic failure.

Benchmarks designed for static datasets do not adequately capture the requirements of real-time applications. For example, scoring with standard classification metrics such as precision and recall do not suffice because they fail to reflect the value of early detection. An artificial separation into training and test sets does not properly capture a streaming scenario nor does it properly evaluate a continuously learning algorithm. The NAB methodology and scoring rules (described below) are designed with such criteria in mind. Through experience with customers and researchers we also discovered it would be beneficial for the industry to include real-world labeled data from multiple domains. Such data is rare and valuable, and NAB attempts to incorporate such a dataset as part of the benchmark. There exist two other time-series data corpuses intended for real-time anomaly detection: the UC-Irvine dataset [3] and a recently released dataset from Yahoo Labs [4]. Neither of these include a scoring system but their data could eventually be incorporated into NAB.

NAB attempts to provide a controlled and repeatable environment of tools to test and measure different anomaly detection algorithms on streaming data. We include in this paper an initial evaluation of four different real-time algorithms. At Numenta we have developed an anomaly detection algorithm based on Hierarchical Temporal Memory (HTM). HTM is a continuous learning system derived from theory of the neocortex [5] and is well suited for real-time applications. The algorithm has proven useful in applications such as monitoring server data, geospatial tracking, stock trading metrics, and social media [6]. We also include comparative results with open source anomaly detection algorithms from Etsy Skyline [7], a popular open source algorithm, and two from Twitter [8]. There are, of course, many algorithms we have not directly tested [2, 9-14]. It is our hope that eventually a wide assortment of algorithms will be independently evaluated and results reported in a manner that is objectively comparable.

In the next section we discuss the two main components of NAB: the scoring system and dataset. We then discuss and analyze NAB scoring results for the above algorithms.

II. NUMENTA ANOMALY BENCHMARK

NAB aims to represent the variety of anomalous data and the associated challenges detectors face in real-world streaming applications. We define anomalies in a data stream to be patterns that do not conform to past patterns of behavior for the stream. This definition encompasses both point anomalies (or spatial anomalies) as well as temporal anomalies. For example, a spiking point anomaly occurs when a single data point extends well above or below the expected range. Streaming data commonly also contains temporal anomalies, such as a change in the frequency, sudden erratic behavior of a metric, or other temporal deviations. Anomalies are defined with respect to past behavior. This means a new behavior can be anomalous at first but ceases to be anomalous if it persists; i.e. a new normal pattern is established. Fig. 1 shows a few representative anomalies taken from the NAB dataset.

In the next two sections we discuss both the NAB dataset and scoring system, and the qualities that make them ideal for evaluating real-world anomaly detection algorithms.

A. *Benchmark Dataset*

In the current version of NAB we focus on time-series data where each row contains a time stamp plus a single scalar value. The requirements are then to (i) include all types of streaming data anomalies, (ii) include a variety of data metrics, and (iii) present common challenges such as noise and establishing new normal patterns.

Anomalous patterns differ significantly across applications. A one-second latency in periodic EKG data could be a significant fluctuation, but the same pattern in stock trading volume may be meaningless. It is thus important for the NAB dataset to include metrics across a variety of domains and applications. The data currently in the NAB corpus represents a variety of metrics ranging from IT metrics such as network utilization to sensors on industrial machines to social media chatter. We also include some artificially-generated data files that test anomalous behaviors not yet represented in the corpus's real data, as well as several data files without any anomalies. The current NAB dataset contains 58 data files, each with 1000-22,000 data instances, for a total of 365,551 data points.

The NAB dataset is labeled by hand, following a meticulous, documented procedure. Labelers must adhere to a set of rules when inspecting data files for anomalies, and a label-combining algorithm formalizes agreement into ground truth labels. The process is designed to mitigate human error as much as possible.[1] In addition a smooth scoring function (described below) ensures that small labeling errors will not cause large changes in reported scores.

It is often prohibitively expensive to collect an accurately labeled set of anomalous data instances that covers all types of anomalous behavior [2]. A key element of the NAB dataset is the inclusion of real-world data with anomalies for which we know the causes. We propose the NAB dataset as a quality collection of time-series data with labeled anomalies, and that it is well suited to be a standard benchmark for streaming applications.

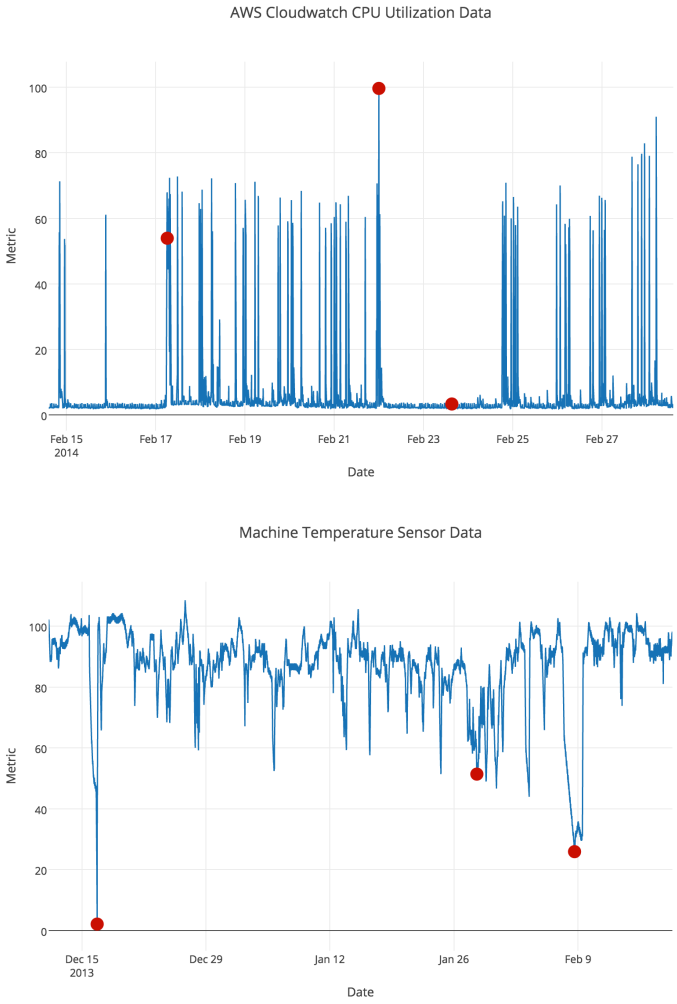


Fig. 1. Two representative examples of real data streams from the NAB dataset. Anomalies are labeled with red circles. The first anomaly in the top figure is subtle and challenging. The spiking behavior does not return to the baseline as expected, and this is soon the new normal pattern. The second anomaly is a simple spike anomaly after which the system returns to previous patterns. The third anomaly identifies a long period inconcsitent with the normal spiking pattern. The bottom figure shows temperature sensor data from an internal component of a large, expensive, industrial machine. The first anomaly was a planned shutdown. The third anomaly is a catastrophic system failure. The second anomaly, a subtle but observable change in the behavior, indicated the actual onset of the problem that led to the eventual system failure.

B. *Scoring Real-Time Anomaly Detectors*

In NAB an anomaly detector accepts data input and outputs instances which it deems to be anomalous. The NAB scoring system formalizes a set of rules to determine the overall quality of anomaly detection. We define the requirements of the ideal, real-world anomaly detector as follows:

i. detects all anomalies present in the streaming data
ii. detects anomalies as soon as possible, ideally before the anomaly becomes visible to a human
iii. triggers no false alarms (no false positives)
iv. works with real time data (no look ahead)

[1] The full labeling process and rules can be found in the NAB wiki, along with the label-combining source code, in the NAB repo [15].

v. is fully automated across all datasets (any data specific parameter tuning must be done online without human intervention)

The NAB scoring algorithm aims to reward these characteristics. There are three key aspects of scoring in NAB: anomaly windows, the scoring function, and application profiles. These are described in more detail below.

Traditional scoring methods, such as precision and recall, don't suffice because they don't effectively test anomaly detection algorithms for real-time use. For example, they do not incorporate time and do not reward early detection. Therefore, the standard classification metrics – true positive (TP), false positive (FP), true negative (TN), and false negative (FN) are not applicable for evaluating algorithms for the above requirements.

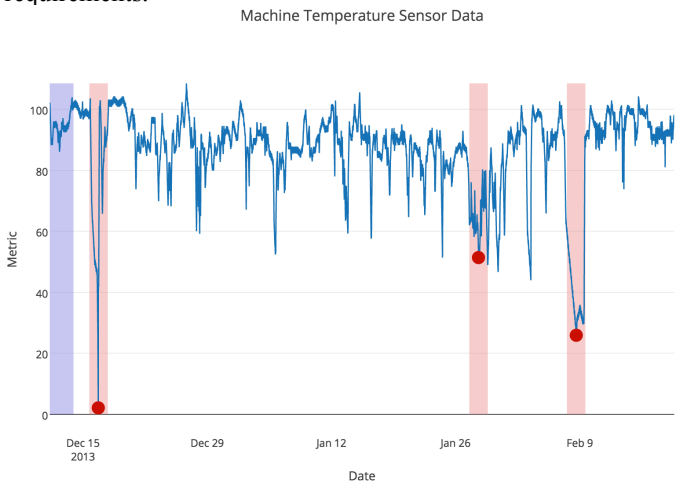


Fig. 2. Shaded red regions represent the anomaly windows for this data file. The shaded purple region is the first 15% of the data file, representing the *probationary period*. During this period the detector is allowed to learn the data patterns without being tested.

To promote early detection NAB defines *anomaly windows*. Each window represents a range of data points that is centered around a ground truth anomaly label. Fig. 2 shows an example using the data from Fig 1. A *scoring function* (described in more detail below) uses these windows to identify and weight true positives, false positives, and false negatives. If there are multiple detections within a window, the earliest detection is given credit and counted as a true positive. Additional positive detections within the window are ignored. The sigmoidal scoring function gives higher positive scores to true positive detections earlier in a window and negative scores to detections outside the window (i.e. the false positives). These properties are illustrated in Fig. 3 with an example.

How large should the windows be? The earlier a detector can reliably identify anomalies the better, implying these windows should be as large as possible. The tradeoff with extremely large windows is that random or unreliable detections would be regularly reinforced. Using the underlying assumption that true anomalies are rare, we define anomaly window length to be 10% the length of a data file, divided by the number of anomalies in the given file. This technique allows us to provide a generous window for early detections and also allow the detector to get partial credit if detections are soon after the ground truth anomaly. 10% is a convenient number but note the exact number is not critical. We tested a range of window sizes (between 5% and 20%) and found that, partly due to the scaled scoring function, the end score was not sensitive to this percentage.

Different applications may place different emphases as to the relative importance of true positives vs. false negatives and false positives. For example, the graph in Fig. 2 represents an expensive industrial machine that one may find in a manufacturing plant. A false negative leading to machine failure in a factory can lead to production outages and be extremely expensive. A false positive on the other hand might require a technician to inspect the data more closely. As such the cost of a false negative is far higher than the cost of a false positive. Alternatively, an application monitoring the statuses of individual servers in a datacenter might be sensitive to the number of false positives and be fine with the occasional missed anomaly since most server clusters are relatively fault tolerant.

To gauge how algorithms operate within these different application scenarios, NAB introduces the notion of *application profiles*. For TPs, FPs, FNs, and TNs, NAB applies different relative weights associated with each profile to obtain a separate score per profile.

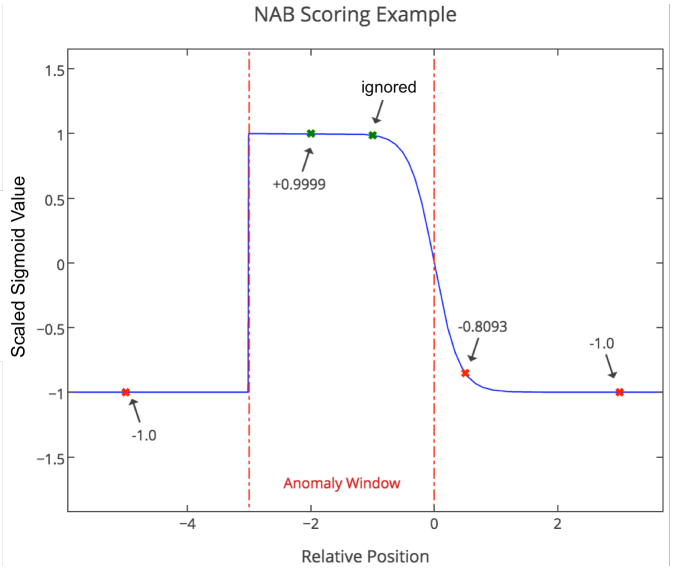


Fig. 3. Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function, the second term in Eq. (1). The first point is an FP preceding the anomaly window (red dashed lines) and contributes -1.0 to the score. Within the window we see two detections, and only count the earliest TP for the score. There are two FPs after the window. The first is less detrimental because it is close to the window, and the second yields -1.0 because it's too far after the window to be associated with the true anomaly. TNs make no score contributions. The scaled sigmoid values are multiplied by the relevant application profile weight, as shown in Eq. (1), the NAB score for this example would calculate as: $-1.0A_{FP} + 0.9999A_{TP} - 0.8093A_{FP} - 1.0A_{FP}$. With the standard application profile this would result in a total score of 0.6909.

NAB includes three different application profiles: standard, reward low FPs, and reward low FNs. The standard profile assigns TPs, FPs, and FNs with relative weights (tied to the window size) such that random detections made 10% of the

time would get a zero final score on average. The latter two profiles accredit greater penalties for FPs and FNs, respectively. These two profiles are somewhat arbitrary but designed to be illustrative of algorithm behavior. The NAB codebase itself is designed such that the user can easily tweak the relative weights and re-score all algorithms. The application profiles thus help evaluate the sensitivity of detectors to specific applications criteria.

The combination of anomaly windows, a smooth temporal scoring function (details in the next section), and the introduction of application profiles allows researchers to evaluate online anomaly detector implementations against the requirements of the ideal detector. Specifically the overall NAB scoring system evaluates real-time performance, prefers earlier detection of anomalies, penalizes "spam" (i.e. FPs), and provides realistic costs for the standard classification evaluation metrics TP, FP, TN, and FN.

C. *Computing NAB Score: Details*

The final NAB score for a given algorithm and a given application profile is computed as follows. Let A be the application profile under consideration, with $A_{TP}, A_{FP}, A_{FN}, A_{TN}$ the corresponding weights for true positives, false positives, etc. These weights are bounded $0 \leq A_{TP}, A_{TN} \leq 1$ and $-1 \leq A_{FP}, A_{FN} \leq 0$. Let D be the set of data files and let Y_d be the set of data instances detected as anomalies for datafile d. (As discussed earlier, we remove redundant detections: if an algorithm produces multiple detections within the anomaly window, we retain only the earliest one.) The number of windows with zero detections in this data file is the number of false negatives, represented by f_d.

The following scaled sigmoidal scoring function defines the weight of individual detections given an anomaly window and the relative position of each detection:

$$\sigma^A(y) = (A_{TP} - A_{FP})\left(\frac{1}{1+e^{5y}}\right) - 1 \quad (1)$$

In Eq. (1), y is the relative position of the detection within the anomaly window. The parameters of Eq. (1) are set such that the right end of the window evaluates to $\sigma(y = 0.0) = 0$ (see Fig. 3), and it yields a max and min of A_{TP} and A_{FP}, respectively. Every detection outside the window is counted as a false positive and given a scaled negative score relative to the preceding window. The function is designed such that detections slightly after the window contribute less negative scores than detections well after the window. Missing a window completely is counted as a false negative and assigned a score of A_{FN}.

The raw score for a data file is the sum of the scores from individual detections plus the impact of missing any windows:

$$S_d^A = \left(\sum_{y \in Y_d} \sigma^A(y)\right) + A_{FN} f_d \quad (2)$$

Eq. (2) accumulates the weighted score for each true positive and false positive, and detriments the total score with a weighted count of all the false negatives. The benchmark raw score for a given algorithm is simply the sum of the raw scores over all the data files in the corpus:

$$S^A = \sum_{d \in D} S_d^A \quad (3)$$

The final reported score is a normalized NAB score computed as follows:

$$S_{NAB}^A = 100 \cdot \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A} \quad (4)$$

Here we scale the score based on the raw scores of a "perfect" detector (one that outputs all true positives and no false positives) and a "null" detector (one that outputs no anomaly detections). It follows from Eq. (4) that the maximum (normalized) score a detector can achieve on NAB is 100, and an algorithm making no detections will score 0.

D. *Other NAB Details*

NAB is most valuable as a community tool, benefitting researchers in academia and industry. In building NAB we have been collaborating with the community, and welcome more contributions of data and additional online anomaly detection algorithms. To this end, NAB is a completely open source code base released under the permissive MIT License [15]. It follows a versioning protocol and public issue tracking, allowing changes and dataset additions to be clearly discussed and communicated. We post scores from contributed algorithms on the NAB scoreboard [16], which reflects a specific NAB version number – NAB v1.0 at the time of paper submission.[1] The codebase is modular and designed to make it easy to test additional algorithms (regardless of programming language), adjust application profiles, and even test custom labeled datafiles.

III. ALGORITHMS TESTED

It is our hope that over time the NAB scoreboard will reflect results from a large number of algorithms. In this paper we report initial NAB results using four open source and commercially used algorithms, plus some control detectors. The four primary algorithms are the Numenta HTM anomaly detector, Etsy Skyline, and two Twitter algorithms, AnomalyDetectionTs and AnomalyDetectionVec. We also use some simple control detectors as baselines. Each of these are briefly described below.

The HTM detector (developed by Numenta and the authors) is based on Hierarchical Temporal Memory (HTM), a machine intelligence technology inspired by the structure of the neocortex [17, 18, 19]. Given a real-time data stream $\cdots, x_{t-2}, x_{t-1}, x_t, x_{t+1}, x_{t+2}, \cdots$, the algorithm models the temporal sequences in that stream. At any point t the HTM makes multiple predictions for x_{t+1}. At time $t+1$ these predictions are compared with the actual values to determine an instantaneous anomaly score between 0 and 1. If any of the predictions are significantly different from x_{t+1} the anomaly score will be 1. Conversely, if one of the predictions is exactly equal to x_{t+1} the anomaly score will be 0. The system

[1] Contributor details can be found in the NAB wiki.

maintains the mean and variance of the recent distribution of anomaly scores. At every time step it outputs the likelihood that the current anomaly score is from the respective normal distribution. The anomaly likelihood score is thresholded to detect the final anomaly.

The Numenta algorithm has several features that make it suitable for real world streaming data. It can deal with both predictable and highly unpredictable data – the final score is the deviation from the typical level of predictability for that stream. The temporal model makes multiple predictions and can thus deal with branching temporal sequences. The algorithm is a continuously learning algorithm, so changes in the statistics of the data are automatically handled without additional retraining. Finally, the system is robust to parameter settings. Thus it is able to handle a wide range of datasets without manual parameter tweaking. The code has been used in commercial applications and is freely available under an AGPL license in Python and C++ [20].

Skyline is a real-time anomaly detection system [7] originally developed by Etsy.com for monitoring its high traffic web site. The algorithm employs a mixture of experts approach. It incorporates a set of simple detectors plus a voting scheme to output the final anomaly score. The detectors include deviation from moving average, deviation from a least squares estimate, deviation from a histogram of past values, etc. Like the Numenta algorithm, Skyline is well suited for analyzing streaming data. The internal estimates are continually adjusted and, due to the mixture of simple experts, it is relatively robust across a wide range of applications. The code is open source and has been tested in commercial settings.

Twitter recently released two versions of a real-time anomaly detection algorithm. It uses a combination of statistical techniques to robustly detect outliers. The Generalized ESD test [21] is combined with robust statistical metrics, and piecewise approximation is used to detect long term trends. One version of the algorithm AnomalyDetectionVec is intended to be more general and detect anomalies in data without timestamps but requires manually tuning of the periodicity. A second version of the algorithm, AnomalyDetectionTs, exploits timestamps to detect periodicity and can detect both short-term (intra-day) and long-term (inter-day) anomalies. Unfortunately AnomalyDetectionTs was unable to calculate the necessary period parameters for some of the NAB data files, and thus we cannot include it in the Table 1 results. It is worth noting that for the data files ADTs successfully ran, it was outperformed by ADVec. Both algorithms have been used in commercial settings and the R code is available as open source [22].

In addition to the above core set of detectors we use three control detectors. A "null" detector outputs a constant anomaly score of 0.5 for all data instances. A "perfect" detector is an oracle that outputs detections that would maximize the NAB score; i.e. it outputs only true positives at the beginning of each window. As discussed earlier the raw NAB scores from these two detectors are used to scale the NAB score for all other algorithms between 0 and 100. We also include a "random" detector that outputs a random anomaly score between 0 and 1 for each data instance. The score from this detector offers some intuition for the chance-level performance on NAB.

Each algorithm in NAB is allowed to output an anomaly score between 0 and 1. The score is then thresholded using a fixed threshold in order to detect an anomaly. NAB includes an automated hill-climbing search for selecting the optimal threshold for each algorithm. This search efficiently optimizes over the range of possible thresholds, where the objective function to be maximized is the NAB scoring function. The detection threshold is thus tuned based on the full NAB dataset, under the constraint that a single fixed number be used for all data files.

IV. RESULTS

A. *Overall NAB Scores*

Table 1 summarizes the scores for all algorithms on the three application profiles using the data files in NAB 1.0. The HTM detector achieves the best overall scores, followed by Etsy and Twitter. Although the HTM detector performs markedly better, the Etsy and Twitter algorithms perform significantly better than chance across all three application profiles.

TABLE I. NAB SCOREBOARD

Detectors	**Scores for Application Profiles**[a]		
	Standard	***Reward low FP***	***Reward low FN***
1. Numenta HTM	64.7	56.5	69.3
2. Twitter ADVec	47.1	33.6	53.5
3. Etsy Skyline	35.7	27.1	44.5
4. Random[b]	16.8	5.8	25.9
5. Null	0.0	0.0	0.0

[a] Each algorithms' parameters were optimized to yield the best NAB scores possible.

[b] Random detections do not yield scores ≈ 0 because of the NAB score optimization step.

B. *Results Analysis*

A detailed analysis of the errors illustrates the primary strengths and weaknesses of each algorithm. Notice all algorithms dropped in score from the Standard to Reward low FP profiles, but why does the Skyline score decrease more than the others? For the NAB corpus of 365,558 data instances, Skyline detects 1161 as anomalous, whereas HTM and ADVec detect 387 and 612, respectively.1 An algorithm that makes many detections is more likely to result in more FPs than one with fewer. Similarly, this algorithm would result in fewer FNs, as reflected in the Reward low FN results, where Skyline increased the most of the tested algorithms.

Investigating individual results files illustrates some of the interesting situations that arise in real time applications and how different detectors behave. Fig. 4 demonstrates the value of continuous learning. This file shows CPU usage on a production server over time and contains two anomalies. The

[1] To maximize the ADVec scores we tuned the "max_anom" and "period" parameters. The former was set to flag a maximum of 0.20% of the data instances in a given file as anomalous. The latter set the period length to 150 data instances (used during seasonal decomposition).

first is a simple spike detected by all algorithms. The second is a sustained shift in the usage. Skyline and HTM both detect the change but then adapt to the new normal (with Skyline adapting quickest). Twitter ADVec however continues to generate anomalies for several days.

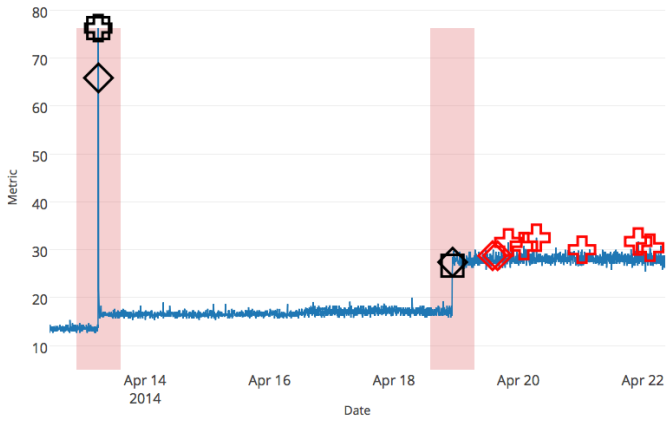


Fig. 4. Detection results for anomalies on a production server based on CPU usage. The shapes correspond to different detectors: HTM, Skyline, and ADVec are diamond, square, and plus respectively. For a given detector, the scored (i.e. the first) TP detection within each window is labeled in black. All FP detections are colored red. As in Fig. 2, the red shaded regions denote the anomaly windows.

Figs. 5 and 6 demonstrate temporal anomalies and their importance in early detection. Fig. 5 shows a close up of the results for the previously discussed machine temperature sensor data file. The first anomaly shown in this plot is a somewhat subtle temporal anomaly where the temporal behavior is unusual but individual readings are within the expected range. This anomaly (which preceded the catastrophic failure on February 8) is only detected by HTM. All three detectors detect the second anomaly, although Skyline and HTM detect it earlier than ADVec. In this plot HTM and Skyline also each have a false positive. Fig. 6 is another example demonstrating early detection. All three detectors detect the anomaly but HTM detects it three hours earlier due to a subtle shift in metric dynamics.

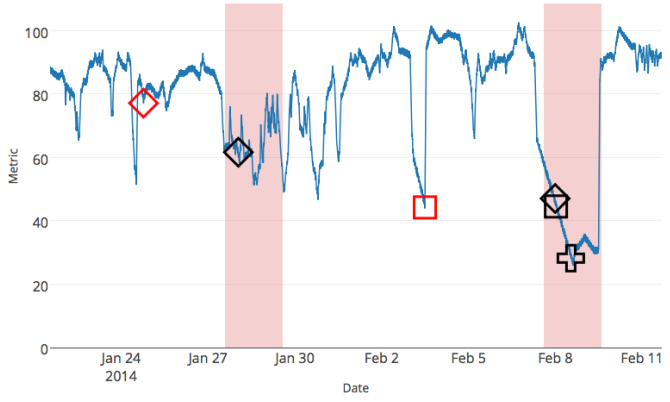


Fig. 5. A close up of the detection results for the machine temperature data of Figs. 1 (bottom) and 2. Only the HTM (diamond) detects the first anomaly (a purely temporal anomaly). All algorithms detect the second anomaly, HTM and Skyline hours before ADVec. HTM and Skyline (diamond) each show an FP.

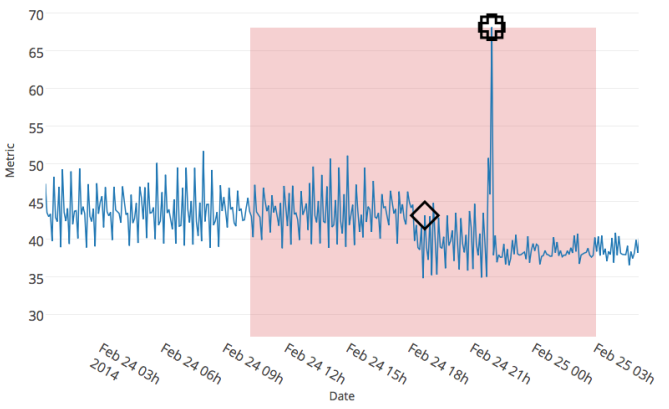


Fig. 6. Detection results demonstrating early detection. Although all detectors correctly identify the anomaly, the HTM (diamond) detects it three hours earlier than Skyline and ADVec due to a change in the metric dynamics.

Both figures illustrate a very common situation that occurs in practice. As demonstrated by the NAB data files, temporal changes in behavior often precede a large easily detectable shift. Temporal and sequence based anomaly detection techniques can thus detect anomalies in streaming data before they are easily visible. This provides hope that such algorithms can be used in production to provide early warning and help avoid catastrophes far more reliably than traditional techniques.

V. Conclusion and Future Work

The aim of NAB is to provide the anomaly detection research community with a controlled and repeatable environment of tools to test and measure different algorithms for real-time streaming applications. The contributions of this work are threefold:

I. Benchmark dataset: real world time-series data files from a variety of domains, labeled with anomalies. Easily accessible real data for streaming applications is rare, and its value for algorithm practitioners is significant. In this paper we have noted some of the insights such as the presence of subtle changes in dynamics preceding large-scale failures.

II. Performance evaluation: a scoring philosophy designed for real time applications. The scoring system in NAB augments traditional metrics by incorporating time, explicitly rewarding early detection. The addition of easily tuned application profiles allows developers to explicitly test algorithms for their specific application requirements.

III. Code library: fully open source repository complete with data, algorithms, and documentation.

We discussed the NAB components and methods, and presented evaluation results for several open source detection algorithms. Although the HTM algorithm outperforms the other tested anomaly detectors, the results show there is still room for improvement. Investigating the results has allowed us to pinpoint the strengths and shortcomings of all algorithms.

We have found NAB to be a comprehensive, robust evaluation tool for real-world anomaly detection algorithms.

NAB v1.0 is ready for researchers to evalute their algorithms and report their results. To move beyond NAB v1.0, future work on NAB will involve adding more real-world data files to the corpus. We also anticipate incorporating multivariate anomaly detection and categorical data. Over time we hope researchers can use NAB to test and develop a large number of anomaly detection algorithms for the specific purpose of applying them to real time streaming applications.

ACKNOWLEDGMENT

We would like to thank Jeff Hawkins, Ian Danforth, Celeste Baranski, Jay Gokhale, and Tom Silver for their contributions to this paper. We also thank the IEEE reviewers for comments that helped improve overall readibility.

REFERENCES

[1] M Tavallaee, N. Stakhanova, and A. Ghorbani, "Toward credible evaluation of anomaly-based intrusion-detection methods," IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, vol. 40, no. 5, pp. 516–524, September 2010.

[2] V. Chandola, A. Banjeree, and V. Kumar, "Anomaly Detection: A Survey," ACM Computing Surveys (CSUR). USA, vol. 41, issue 3, pp 1–72, September 2009.

[3] E. Keogh, J. Lin, and A. Fu, "HOT SAX: Efficiently finding the most unusual time series subsequence," in proceedings of Fifth IEEE International Conference on Data Mining, pp. 226-233, November 2005.

[4] N. Laptev, A. Amizadeh, and Y. Billawala. (2015, March 25) *Yahoo Labs News: Announcing A Benchmark Dataset For Time Series Anomaly Detection* [Online blog]. Available: http://labs.yahoo.com/news/announcing-a-benchmark-dataset-for-time-series-anomaly-detection

[5] J. Hawkins, S. Ahmad, and D. Dubinsky. (2010) *Hierarchical temporal memory including HTM cortical learning algorithms* [Online technical report]. Palo Alto, CA: Numenta, Inc. Available: http://numenta.org/cla-white-paper.html

[6] (2015) *Numenta Applications* [Online website]. Redwood City, CA: Numenta, Inc. Available: http://numenta.com/#applications

[7] A. Stanway. (2013) *etsy/skyline* [Online code repository]. Available: https://github.com/etsy/skyline

[8] A. Kejariwal. (2015, January 6) *Twitter Engineering: Introducing practical and robust anomaly detection in a time series* [Online blog]. Available: https://blog.twitter.com/2015/introducing-practical-and-robust-anomaly-detection-in-a-time-series

[9] F. Serdio and E. Lughofer and K. Pichler and M.Pichler and T. Buchegger and H. Efendic. (2014) "Fault Detection in Multi-Sensor Networks based on Multivariate Time-Series Models and Orthogonal Transformations", Information Fusion, vol. 20, pp. 272-291

[10] N. Laptev, S. Amizadeh, and I. Flint. (2015) *Generic and Scalable Framework for Automated Time-series Anomaly Detection*. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1939–1947.

[11] U. Rebbapragada, P. Protopapas, C. E. Brodley, and C. Alcock. (2009) "Finding anomalous periodic time series : An application to catalogs of periodic variable stars," Machine Learning, vol. 74, no. 3, pp. 281–313.

[12] V. Chandola, V. Mithal, and V. Kumar. (2008) "Comparative evaluation of anomaly detection techniques for sequence data." Eighth IEEE International Conference on Data Mining.

[13] D. Dasgupta, and S. Forrest. (1996) "Novelty detection in time series data using ideas from immunology." Proceedings of the International Conference on Intelligent Systems.

[14] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. (2003) "A symbolic representation of time series, with implications for streaming algorithms." Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. ACM.

[15] Numenta, Inc. (2015) *NAB: Numenta Anomaly Benchmark* [Online code repository]. Redwood City, CA: Numenta, Inc. Available: https://github.com/numenta/NAB

[16] A. Lavin. (2015) *NAB Scoreboard* [Online wiki]. Redwood City, CA: Numenta, Inc. Available: https://github.com/numenta/NAB/wiki/NAB%20Scoreboard

[17] J. Hawkins and S. Blakeslee, *On Intelligence*. New York: Henry Holt and Company, 2004.

[18] J. Hawkins, S. Ahmad, and D. Dubinsky. (2014) *The Science of Anomaly Detection* [Online technical report]. Redwood City, CA: Numenta, Inc. Available: http://numenta.com/#technology

[19] S. Purdy. (2014, October 17) *Science of Anomaly Detection* [Online video]. Redwood City, CA: Numenta, Inc. Available: http://numenta.com/learn/science-of-anomaly-detection.html

[20] Numenta, Inc. (2015) *NuPIC: Numenta Platform for Intelligent Computing* [Online code repository]. Redwood City, CA: Numenta, Inc. Available: https://github.com/numenta/nupic

[21] B. Rosner, "Percentage points for a generalized ESD many-outlier procedure", Technometrics, 25(2), pp. 165-172, May 1983.

[22] A. Kejariwal. (2015) *Twitter/AnomalyDetection* [Online code repository]. Available: https://github.com/twitter/AnomalyDetection

Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex

Jeff Hawkins*, Subutai Ahmad

Numenta, Inc, Redwood City, California, United States of America

*Corresponding author
Emails: jhawkins@numenta.com, sahmad@numenta.com

Keywords: neocortex, prediction, neocortical theory, active dendrites, sequence memory.

A version of this manuscript has been submitted for publication as of October 30, 2015. ***Note that figures and tables are at the end of this PDF.***

Please contact the authors for updated citation information.

Abstract

Neocortical neurons have thousands of excitatory synapses. It is a mystery how neurons integrate the input from so many synapses and what kind of large-scale network behavior this enables. It has been previously proposed that non-linear properties of dendrites enable neurons to recognize multiple patterns. In this paper we extend this idea by showing that a neuron with several thousand synapses arranged along active dendrites can learn to accurately and robustly recognize hundreds of unique patterns of cellular activity, even in the presence of large amounts of noise and pattern variation. We then propose a neuron model where some of the patterns recognized by a neuron lead to action potentials and define the classic receptive field of the neuron, whereas the majority of the patterns recognized by a neuron act as predictions by slightly depolarizing the neuron without immediately generating an action potential. We then present a network model based on neurons with these properties and show that the network learns a robust model of time-based sequences. Given the similarity of excitatory neurons throughout the neocortex and the importance of sequence memory in inference and behavior, we propose that this form of sequence memory is a universal property of neocortical tissue. We further propose that cellular layers in the neocortex implement variations of the same sequence memory algorithm to achieve different aspects of inference and behavior. The neuron and network models we introduce are robust over a wide range of parameters as long as the network uses a sparse distributed code of cellular activations. The sequence capacity of the network scales linearly with the number of synapses on each neuron. Thus neurons need thousands of synapses to learn the many temporal patterns in sensory stimuli and motor sequences.

1. Introduction

Excitatory neurons in the neocortex have thousands of excitatory synapses. The proximal synapses, those closest to the cell body, have a relatively large effect on the likelihood of a cell generating an action potential. However, a majority of the synapses are distal, or far from the cell body. The activation of a single distal synapse has little effect at the soma, and for many years it was hard to imagine how the thousands of distal synapses could play an important role in determining a cell's responses (Major et al., 2013). We now know that dendrite branches are active processing elements. The activation of several distal synapses within close spatial and temporal proximity can lead to a local dendritic NMDA spike and consequently a significant and sustained depolarization of the soma (Antic et al., 2010; Major et al., 2013). This has led some researchers to suggest that dendritic branches act as independent pattern recognizers (Poirazi et al., 2003; Polsky et al., 2004). Yet, despite the many advances in understanding the active properties of dendrites, it remains a mystery why neurons have so many synapses and what their precise role is in memory and cortical processing.

Lacking a theory of why neurons have active dendrites, almost all artificial neural networks, such as those used in deep learning (LeCun et al., 2015) and spiking neural networks (Maass, 1997), use artificial neurons without active dendrites and with unrealistically few synapses, strongly suggesting they are missing key functional aspects of real neural tissue. If we want to understand how the neocortex works and build systems that work on the same principles as the neocortex, we need an understanding of how biological neurons use their thousands of synapses and active dendrites. Of course, neurons cannot be understood in isolation. We also need a complementary theory of how networks of neurons, each with thousands of synapses, work together towards a common purpose.

In this paper we introduce such a theory. First, we show how a typical pyramidal neuron with active dendrites and thousands of synapses can recognize hundreds of unique patterns of cellular activity. We show that a neuron can recognize hundreds of patterns even in the presence of large amounts of noise and variability as long as overall neural activity is sparse. Next we introduce a neuron model where the inputs to different parts of the dendritic tree serve different purposes. In this model the patterns recognized by a neuron's distal synapses are used for prediction. Each neuron learns to recognize hundreds of patterns that often precede the cell becoming active. The recognition of any one of these learned patterns acts as a prediction by depolarizing the cell without directly causing an action potential. Finally, we show how a network of neurons with this property will learn and recall sequences of patterns. The network model relies on depolarized neurons firing quickly and inhibiting other nearby neurons, thus biasing the network's activation towards its predictions. Through simulation we illustrate that the sequence memory network exhibits numerous desirable properties such as on-line learning, multiple simultaneous predictions, and robustness.

Given the similarity of neurons throughout the neocortex and the importance of sequence memory for inference and behavior, we propose that sequence memory is a property of neural tissue throughout the neocortex and thus represents a new and important unifying principle for understanding how the neocortex works.

2. Results

2.1. Neurons Recognize Multiple Patterns

It is common to think of a neuron as recognizing a single pattern of activity on its synapses. This notion, sometimes called a "point neuron", forms the basis of almost all artificial neural networks (Fig. 1A).

[Figure 1 about here – see end of manuscript]

Active dendrites suggest a different view of the neuron, where neurons recognize many unique patterns (Larkum and Nevian, 2008; Poirazi et al., 2003; Polsky et al., 2004). Experimental results show that the coincident activation of eight to twenty synapses in close spatial proximity on a dendrite will combine in a non-linear fashion and cause an NMDA dendritic spike (Larkum et al., 1999; Major et al., 2013; Schiller and Schiller, 2001; Schiller et al., 2000). Thus, a small set of neighboring synapses acts as a pattern detector. It follows that the thousands of synapses on a cell's dendrites

act as a set of independent pattern detectors. The detection of any of these patterns causes an NMDA spike and subsequent depolarization at the soma.

It might seem that eight to twenty synapses could not reliably recognize a pattern of activity in a large population of cells. However, robust recognition is possible if the patterns to be recognized are sparse; i.e. few neurons are active relative to the population (Olshausen and Field, 2004). For example, consider a population of 200K cells where 1% (2,000) of the cells are active at any point in time. We want a neuron to detect when a particular pattern occurs in the 200K cells. If a section of the neuron's dendrite forms new synapses to just 10 of the 2,000 active cells, and the threshold for generating an NMDA spike is 10, then the dendrite will detect the target pattern when all 10 synapses receive activation at the same time. Note that the dendrite could falsely detect many other patterns that share the same 10 active cells. However, if the patterns are sparse, the chance that the 10 synapses would become active for a different random pattern is small. In this example it is only 9.8×10^{-21}.

The probability of a false match can be calculated precisely as follows. Let n represent the size of the cell population and a the number of active cells in that population at a given point in time, for sparse patterns $a \ll n$. Let s be the number of synapses on a dendritic segment and θ be the NMDA spike threshold. We say the segment recognizes a pattern if at least θ synapses become active, i.e. at least θ of the s synapses match the currently active cells.

Assuming a random distribution of patterns, the exact probability of a false match is given by:

$$\frac{\sum_{b=\theta}^{s} \binom{s}{b} \times \binom{n-s}{a-b}}{\binom{n}{a}} \quad (1)$$

The denominator is simply the total number of possible patterns containing a active cells in a population of n total cells. The numerator counts the number of patterns that would connect to θ or more of the s synapses on one dendritic segment. A more detailed description of this equation can be found in (Ahmad and Hawkins, 2015).

The equation shows that a non-linear dendritic segment can robustly classify a pattern by sub-sampling (forming synapses to only a small number of the cells in the pattern to be classified). Table A in S1 Text lists representative error probabilities calculated from Eq. (1).

By forming more synapses than necessary to generate an NMDA spike, recognition becomes robust to noise and variation. For example, if a dendrite has an NMDA spike threshold of 10, but forms 20 synapses to the pattern it wants to recognize, twice as many as needed, it allows the dendrite to recognize the target pattern even if 50% of the cells are changed or inactive. The extra synapses also increase the likelihood of a false positive error. Although the chance of error has increased, Eq. (1) shows that it is still tiny when the patterns are sparse. In the above example, doubling the number of synapses and hence introducing a 50% noise tolerance, increases the chance of error to only 1.6×10^{-18}. Table 1B in S1 Text lists representative error rates when the number of synapses exceeds the threshold.

The synapses recognizing a given pattern have to be co-located on a dendritic segment. If they lie within 40μm of each other then as few as eight synapses are sufficient to create an NMDA spike (Major et al., 2008). If the synapses are spread out along the dendritic segment, then up to twenty synapses are needed (Major et al., 2013). A dendritic segment can contain several hundred synapses; therefore each segment can detect multiple patterns. If synapses that recognize different patterns are mixed together on the dendritic segment, it introduces an additional possibility of error by co-activating synapses from different patterns. The probability of this type of error depends on how many sets of synapses share the dendritic segment and the sparsity of the patterns to be recognized. For a wide range of values the chance for this type of error is still low (Table C in S1 Text). Thus the placement of synapses to recognize a particular pattern is somewhat precise (they must be on the same dendritic segment and ideally within 40μm of each other), but also somewhat imprecise (mixing with other synapses is unlikely to cause errors).

If we assume an average of 20 synapses are allocated to recognize each pattern, and that a neuron has 6,000 synapses, then a cell would have the ability to recognize approximately 300 different patterns. This is a rough approximation, but makes evident that a neuron with active dendrites can learn to reliably recognize hundreds of patterns within a large population of cells. The recognition of any one of these patterns will depolarize the cell. Since all excitatory neurons in the neocortex have thousands of synapses, and, as far as we know, they all have active dendrites, then each and every excitatory neocortical neuron recognizes hundreds of patterns of neural activity.

In the next section we propose that most of the patterns recognized by a neuron do not directly lead to an action potential, but instead play a role in how networks of neurons make predictions and learn sequences.

2.1.1. Three Sources of Synaptic Input to Cortical Neurons

Neurons receive excitatory input from different sources that are segregated on different parts of the dendritic tree. Fig. 1B shows a typical pyramidal cell, the most common excitatory neuron in the neocortex. We show the input to the cell divided into three zones. The proximal zone receives feedforward input. The basal zone receives contextual input, mostly from nearby cells in the same cortical region (Petreanu et al., 2009; Rah et al., 2013; Yoshimura et al., 2000). The apical zone receives feedback input (Spruston, 2008). (The second most common excitatory neuron in the neocortex is the spiny stellate cell; we suggest they be considered similar to pyramidal cells minus the apical dendrites.) We propose the three zones of synaptic integration on a neuron (proximal, basal, and apical) serve the following purposes.

Proximal Synapses Define the Classic Receptive Field of a Cell

The synapses on the proximal dendrites (typically several hundred) have a relatively large effect at the soma and

therefore are best situated to define the basic receptive field response of the neuron (Spruston, 2008). If the coincident activation of a subset of the proximal synapses is sufficient to generate a somatic action potential and If the inputs to the proximal synapses are sparsely active, then the proximal synapses will recognize multiple unique feedforward patterns in the same manner as discussed earlier. Therefore, the feedforward receptive field of a cell can be thought of as a union of feedforward patterns.

Basal Synapses Learn Transitions in Sequences

We propose that basal dendrites of a neuron recognize patterns of cell activity that precede the neuron firing, in this way the basal dendrites learn and store transitions between activity patterns. When a pattern is recognized on a basal dendrite it generates an NMDA spike. The depolarization due to an NMDA spike attenuates in amplitude by the time it reaches the soma, therefore when a basal dendrite recognizes a pattern it will depolarize the soma but not enough to generate a somatic action potential (Antic et al., 2010; Major et al., 2013). We propose this sub-threshold depolarization is an important state of the cell. It represents a prediction that the cell will become active shortly and plays an important role in network behavior. A slightly depolarized cell fires earlier than it would otherwise if it subsequently receives sufficient feedforward input. By firing earlier it inhibits neighboring cells, creating highly sparse patterns of activity for correctly predicted inputs. We will explain this mechanism more fully in a later section.

Apical Synapses Invoke a Top-down Expectation

The apical dendrites of a neuron also generate NMDA spikes when they recognize a pattern (Cichon and Gan, 2015). An apical NMDA spike does not directly affect the soma. Instead it can lead to a Ca2+ spike in the apical dendrite (Golding et al., 1999; Larkum et al., 2009). A single apical Ca2+ spike will depolarize the soma, but typically not enough to generate a somatic action potential (Antic et al., 2010). The interaction between apical Ca2+ spikes, basal NMDA spikes, and somatic action potentials is an area of ongoing research (Larkum, 2013), but we can say that under many conditions a recognized pattern on an apical dendrite will depolarize the cell and therefore have a similar effect as a recognized pattern on a basal dendrite. We propose that the depolarization caused by the apical dendrites is used to establish a top-down expectation, which can be thought of as another form of prediction.

2.1.2. The HTM Model Neuron

Fig. 1C shows an abstract model of a pyramidal neuron we use in our software simulations. We model a cell's dendrites as a set of threshold coincidence detectors; each with its own synapses. If the number of active synapses on a dendrite/coincidence detector exceeds a threshold the cell detects a pattern. The coincidence detectors are in three groups corresponding to the proximal, basal, and apical dendrites of a pyramidal cell. We refer to this model neuron as an "HTM neuron" to distinguish it from biological neurons and point neurons. HTM is an acronym for Hierarchical Temporal Memory, a term used to describe our models of neocortex (Hawkins et al., 2011). HTM neurons used in the simulations for this paper have 128 dendrite/coincidence detectors with up to 40 synapses per dendrite. For clarity, Fig. 1C shows only a few dendrites and synapses.

2.2. Networks of Neurons Learn Sequences

Because all tissue in the neocortex consists of neurons with active dendrites and thousands of synapses, it suggests there are common network principles underlying everything the neocortex does. This leads to the question, what network property is so fundamental that it is a necessary component of sensory inference, prediction, language, and motor planning?

We propose that the most fundamental operation of all neocortical tissue is learning and recalling sequences of patterns (Hawkins and Blakeslee, 2004), what Karl Lashley famously called "the most important and also the most neglected problem of cerebral physiology" (Lashley, 1951). More specifically, we propose that each cellular layer in the neocortex implements a variation of a common sequence memory algorithm. We propose cellular layers use sequence memory for different purposes, which is why cellular layers vary in details such as size and connectivity. In this paper we illustrate what we believe is the basic sequence memory algorithm without elaborating on its variations.

We started our exploration of sequence memory by listing several properties required of our network in order to model the neocortex.

1) On-line learning
Learning must be continuous. If the statistics of the world change, the network should gradually and continually adapt with each new input.

2) High-order predictions
Making correct predictions with complex sequences requires the ability to incorporate contextual information from the past. The network needs to dynamically determine how much temporal context is needed to make the best predictions. The term "high-order" refers to "high-order Markov chains" which have this property.

3) Multiple simultaneous predictions
Natural data streams often have overlapping and branching sequences. The sequence memory therefore needs to make multiple predictions at the same time.

4) Local learning rules
The sequence memory must only use learning rules that are local to each neuron. The rules must be local in both space and time, without the need for a global objective function.

5) Robustness
The memory should exhibit robustness to high levels of noise, loss of neurons, and natural variation in the input. Degradation in performance under these conditions should be gradual.

All these properties must occur simultaneously in the context of continuously streaming data.

2.2.1. *Mini-columns and Neurons: Two Representations*

High-order sequence memory requires two simultaneous representations. One represents the feedforward input to the network and the other represents the feedforward input in a particular temporal context. To illustrate this requirement, consider two abstract sequences "ABCD" and "XBCY", where each letter represents a sparse pattern of activation in a population of neurons. Once these sequences are learned the network should predict "D" when presented with sequence "ABC" and it should predict "Y" when presented with sequence "XBC". Therefore, the internal representation during the subsequence "BC" must be different in the two cases; otherwise the correct prediction can't be made after "C" is presented.

Fig. 2 illustrates how we propose these two representations are manifest in a cellular layer of cortical neurons. The panels in Fig. 2 represent a slice through a single cellular layer in the neocortex (Fig. 2A). The panels are greatly simplified for clarity. Fig. 2B shows how the network represents two input sequences before the sequences are learned. Fig. 2C shows how the network represents the same input after the sequences are learned. Each feedforward input to the network is converted into a sparse set of active mini-columns. (Mini-columns in the neocortex span multiple cellular layers. Here we are only referring to the cells in a mini-column in one cellular layer.) All the neurons in a mini-column share the same feedforward receptive fields. If an unanticipated input arrives, then all the cells in the selected mini-columns will recognize the input pattern and become active. However, in the context of a previously learned sequence, one or more of the cells in the mini-columns will be depolarized. The depolarized cells will be the first to generate an action potential, inhibiting the other cells nearby. Thus a predicted input will lead to a very sparse pattern of cell activation that is unique to a particular element, at a particular location, in a particular sequence.

[Figure 2 about here – see end of manuscript]

2.2.2. *Basal Synapses Are the Basis of Sequence Memory*

In this theory, cells use their basal synapses to learn the transitions between input patterns. With each new feedforward input some cells become active via their proximal synapses. Other cells, using their basal synapses, learn to recognize this active pattern and upon seeing the pattern again, become depolarized, thereby predicting their own feedforward activation in the next input. Feedforward input activates cells, while basal input generates predictions. As long as the next input matches the current prediction, the sequence continues, Fig. 3. Fig. 3A shows both active cells and predicted cells while the network follows a previously learned sequence.

[Figure 3 about here – see end of manuscript]

Often the network will make multiple simultaneous predictions. For example, suppose that after learning the sequences "ABCD" and "XBCY" we expose the system to just the ambiguous sub-sequence "BC". In this case we want the system to simultaneously predict both "D" and "Y". Fig. 3B illustrates how the network makes multiple predictions when the input is ambiguous. The number of simultaneous predictions that can be made with low chance of error can again be calculated via Eq. (1). Because the predictions tend to be highly sparse, it is possible for a network to predict dozens of patterns simultaneously without confusion. If an input matches any of the predictions it will result in the correct highly-sparse representation. If an input does not match any of the predictions all the cells in a column will become active, indicating an unanticipated input.

Although every cell in a mini-column shares the same feedforward response, their basal synapses recognize different patterns. Therefore cells within a mini-column will respond uniquely in different learned temporal contexts, and overall levels of activity will be sparser when inputs are anticipated. Both of these attributes have been observed (Martin and Schröder, 2013; Vinje and Gallant, 2002; Yen et al., 2007).

For one of the cells in the last panel of Fig. 3A, we show three connections the cell used to make a prediction. In real neurons, and in our simulations, a cell would form 15 to 40 connections to a subset of a larger population of active cells.

2.2.3. *Apical Synapses Create a Top-Down Expectation*

Feedback axons between neocortical regions often form synapses (in layer 1) with apical dendrites of pyramidal neurons whose cell bodies are in layers 2, 3, and 5. It has long been speculated that these feedback connections implement some form of expectation or bias (Lamme et al., 1998). Our neuron model suggests a mechanism for top-down expectation in the neocortex. Fig. 4 shows how a stable feedback pattern to apical dendrites can predict multiple elements in a sequence all at the same time. When a new feedforward input arrives it will be interpreted as part of the predicted sequence. The feedback biases the input towards a particular interpretation. Again, because the patterns are sparse, many patterns can be simultaneously predicted.

[Figure 4 about here – see end of manuscript]

Thus there are two types of prediction occurring at the same time. Lateral connections to basal dendrites predict the next input, and top-down connections to apical dendrites predict multiple sequence elements simultaneously. The physiological interaction between apical and basal dendrites is an area of active research (Larkum, 2013) and will likely lead to a more nuanced interpretation of their roles in inference and prediction. However, we propose that the mechanisms shown in Figs. 2, 3 and 4 are likely to continue to play a role in that final interpretation.

2.2.4. *Synaptic Learning Rule*

Our neuron model requires two changes to the learning rules by which most neural models learn. First, learning occurs by growing and removing synapses from a pool of "potential" synapses (Chklovskii et al., 2004). Second, Hebbian learning and synaptic change occur at the level of the dendritic segment, not the entire neuron (Stuart and Häusser, 2001).

Potential Synapses

For a neuron to recognize a pattern of activity it requires a set of co-located synapses (typically fifteen to twenty) that connect to a subset of the cells that are active in the pattern to

be recognized. Learning to recognize a new pattern is accomplished by the formation of a set of new synapses collocated on a dendritic segment.

Figure 5 shows how we model the formation of new synapses in a simulated HTM neuron. For each dendritic segment we maintain a set of "potential" synapses between the dendritic segment and other cells in the network that could potentially form a synapse with the segment (Chklovskii et al., 2004). The number of potential synapses is larger than the number of actual synapses. We assign each potential synapse a scalar value called "permanence" which represents stages of growth of the synapse. A permanence value close to zero represents an axon and dendrite with the potential to form a synapse but that have not commenced growing one. A 1.0 permanence value represents an axon and dendrite with a large fully formed synapse.

[Figure 5 about here – see end of manuscript]

The permanence value is incremented and decremented using a Hebbian-like rule. If the permanence value exceeds a threshold, such as 0.3, then the weight of the synapse is 1, if the permanence value is at or below the threshold then the weight of the synapse is 0. The threshold represents the establishment of a synapse, albeit one that could easily disappear. A synapse with a permanence value of 1.0 has the same effect as a synapse with a permanence value at threshold but is not as easily forgotten. Using a scalar permanence value enables on-line learning in the presence of noise. A previously unseen input pattern could be noise or it could be the start of a new trend that will repeat in the future. By growing new synapses, the network can start to learn a new pattern when it is first encountered, but only act differently after several presentations of the new pattern. Increasing permanence beyond the threshold means that patterns experienced more than others will take longer to forget.

HTM neurons and HTM networks rely on distributed patterns of cell activity, thus the activation strength of any one neuron or synapse is not very important. Therefore, in HTM simulations we model neuron activations and synapse weights with binary states. Additionally, it is well known that biological synapses are stochastic (Faisal et al., 2008), so a neocortical theory cannot require precision of synaptic efficacy. Although scalar states and weights might improve performance, they are not required from a theoretical point of view and all of our simulations have performed well without them. The formal learning rules used in our HTM network simulations are presented in the Materials and Methods section.

3. Simulation Results

Fig. 6 illustrates the performance of a network of HTM neurons implementing a high-order sequence memory. The network used in Fig. 6 consists of 2048 mini-columns with 32 neurons per mini-column. Each neuron has 128 basal dendritic segments, and each dendritic segment has up to 40 actual synapses. Because this simulation is designed to only illustrate properties of sequence memory it does not include apical synapses. The network exhibits all five of the desired properties for sequence memory listed earlier.

[Figure 6 about here – see end of manuscript]

Although we have applied HTM networks to many types of real-world data, in Fig. 6 we use an artificial data set to more clearly illustrate the network's properties. The input is a stream of elements, where every element is converted to a 2% sparse activation of mini-columns (40 active columns out of 2048 total). The network learns a predictive model of the data based on observed transitions in the input stream. In Fig. 6 the data stream fed to the network contains a mixture of random elements and repeated sequences. The embedded sequences are six elements long and require high-order temporal context for full disambiguation and best prediction accuracy, e.g. "XABCDE" and "YABCFG". For this simulation we designed the input data stream such that the maximum possible average prediction accuracy is 50% and this is only achievable by using high-order representations.

Fig. 6A illustrates on-line learning and high-order predictions. The prediction accuracy of the HTM network over time is shown in red. The prediction accuracy starts at zero and increases as the network discovers the repeated temporal patterns mixed within the random transitions. For comparison, the accuracy of a first-order network (created by using only one cell per column) is shown in blue. After sufficient learning, the high-order HTM network achieves the maximum possible prediction accuracy of 50% whereas the first-order network only achieves about 33% accuracy. After the networks reached their maximum performance the embedded sequences were modified. The accuracy drops at that point, but since the network is continually learning it recovers by learning the new high-order patterns.

Fig. 6B illustrates the robustness of the network. After the network reached stable performance we inactivated a random selection of neurons. At up to about 40% cell death there was minimal impact on performance. This robustness is due to the noise tolerance described earlier that occurs when a dendritic segment forms more synapses than necessary to generate an NMDA spike. At higher levels of cell death the network performance initially declines but then recovers as the network relearns the patterns using the remaining neurons.

4. Discussion

We presented a model cortical neuron that is substantially different than model neurons used in most artificial neural networks. The key feature of the model neuron is its use of active dendrites and thousands of synapses, allowing the neuron to recognize hundreds of unique patterns in large populations of cells. We showed that a neuron can reliably recognize many patterns, even in the presence of large amounts of noise and variation. In this model, proximal synapses define the feedforward receptive field of a cell. The basal and apical synapses depolarize the cell, representing predictions.

We showed that a network of these neurons will learn a predictive model of a stream of data. Basal synapses detect contextual patterns that predict the next feedforward input. Apical synapses detect feedback patterns that predict entire sequences. The operation of the neuron and the network rely on neural activity being sparse. The sequence memory model learns continuously, uses variable amounts of context to make predictions, makes multiple simultaneous predictions,

relies on local learning rules, and is robust to failure of network elements, noise, and variation.

Although we refer to the network model as a "sequence memory", it is actually a memory of transitions. There is no representation or concept of the length of sequences or of the number of stored sequences. The network only learns transitions between inputs. Therefore, the capacity of a network is measured by how many transitions a given network can store. This can be calculated as the product of the expected duty cycle of an individual neuron (cells per column/column sparsity) times the number of patterns each neuron can recognize on its basal dendrites. For example, a network where 2% of the columns are active, each column has 32 cells, and each cell recognizes 200 patterns on its basal dendrites, can store approximately 320,000 transitions ((32/0.02)*200). The capacity scales linearly with the number of cells per column and the number of patterns recognized by the basal synapses of each neuron.

Another important capacity metric is how many times a particular input can appear in different temporal contexts without confusion. This is analogous to how many times a particular musical interval can appear in melodies without confusion, or how many times a particular word can be memorized in different sentences. If mini-columns have 32 cells it doesn't mean a particular pattern can have only 32 different representations. For example, if we assume 40 active columns per input, 32 cells per column, and one active cell per column, then there are 32^{40} possible representations of each input pattern, a practically unlimited number. Therefore, the practical limit is not representational but memory-based. The capacity is determined by how many transitions can be learned with a particular sparse set of columns.

So far we have only discussed cellular layers where all cells in the network can potentially connect to all other cells with equal likelihood. This works well for small networks but not for large networks. In the neocortex, it is well known that most regions have a topological organization. For example cells in region V1 receive feedforward input from only a small part of the retina and receive lateral input only from a local area of V1. HTM networks can be configured this way by arranging the columns in a 2D array and selecting the potential synapses for each dendrite using a 2D probability distribution centered on the neuron. Topologically organized networks can be arbitrarily large.

There are several testable predictions that follow from this theory.

1) The theory provides an algorithmic explanation for the experimentally observed phenomenon that overall cell activity becomes sparser during a continuous predictable sensory stream (Martin and Schröder, 2013; Vinje and Gallant, 2002; Yen et al., 2007). In addition, it predicts that unanticipated inputs will result in higher cell activity, which should be correlated vertically within mini-columns. Anticipated inputs on the other hand will result in activity that is uncorrelated within mini-columns. It is worth noting that mini-columns are not a strict requirement of this theory. The model only requires the presence of small groups of cells that share feedforward responses and that are mutually inhibitory. We refer to these groups as mini-columns, but the columnar aspect is not a requirement, and the groupings could be independent of actual mini-columns.

2) A second core prediction of the theory is that the current pattern of cell activity contains information about past stimuli. Early experimental results supporting this prediction have been reported in (Nikolić et al., 2009). Further studies are required to validate the exact nature of dynamic cell activity and the role of temporal context in high order sequences.

3) Synaptic plasticity should be localized to dendritic segments that have been depolarized via synaptic input followed a short time later by a back action potential. This effect has been reported (Losonczy et al., 2008), though the phenomenon has yet to be widely established.

4) There should be few, ideally only one, excitatory synapses formed between a given axon and a given dendritic segment. If an excitatory axon made many synapses in close proximity onto a single dendrite then the presynaptic cell would dominate in causing an NMDA spike. Two, three, or even four synapses from a single axon onto a single dendritic segment could be tolerated, but if axons routinely made more synapses to a single dendritic segment it would lead to errors. Pure Hebbian learning would seem to encourage forming multiple synapses. To prevent this from happening we predict the existence of a mechanism that actively discourages the formation of a multiple synapses after one has been established. An axon can form synapses onto different dendritic segments of the same neuron without causing problems, therefore we predict this mechanism will be spatially localized within dendritic segments or to a local area of an axonal arbor.

5) When a cell depolarized by an NMDA spike subsequently generates an action potential via proximal input, it needs to inhibit all other nearby excitatory cells. This requires a fast, probably single spike, inhibition. Fast-spiking basket inhibitory cells are the most likely source for this rapid inhibition (Hu et al., 2014).

6) All cells in a mini-column need to learn common feedforward responses. This requires a mechanism to encourage all the cells in a mini-column to become active simultaneously while learning feedforward patterns. This requirement for mutual excitation seems at odds with the prior requirement for mutual inhibition when one or more cells are slightly depolarized. We don't have a specific proposal for how these two requirements are met but we predict a mechanism where sometimes cells in a column are mutually excited and at other times they are mutually inhibited.

Pyramidal neurons are common in the hippocampus. Hence, parts of our neuron and network models might apply to the hippocampus. However, the hippocampus is known for fast learning, which is incompatible with growing new synapses, as synapse formation can take hours in an adult (Holtmaat and Svoboda, 2009; Knott et al., 2002; Niell et al., 2004; Trachtenberg et al., 2002). Rapid learning could be achieved in our model if instead of growing new synapses, a cell had a multitude of inactive, or "silent" synapses (Kerchner and Nicoll, 2008). Rapid learning would then occur by turning silent synapses into active synapses. The downside of this approach is a cell would need many more synapses, which is

metabolically expensive. Pyramidal cells in hippocampal region CA2 have several times the number of synapses as pyramidal cells in neocortex (Megías et al., 2001). If most of these synapses were silent it would be evidence to suggest that region CA2 is also implementing a variant of our proposed sequence memory.

It is instructive to compare our proposed biological sequence memory mechanism to other sequence memory techniques used in the field of machine learning. The most common technique is Hidden Markov Models (HMMs) (Rabiner and Juang, 1986). HMMs are widely applied, particularly in speech recognition. The basic HMM is a first-order model and its accuracy would be similar to the first-order model shown in Fig. 6A. Variations of HMMs can model restricted high order sequences by encoding high-order states by hand. More recently, recurrent neural networks, specifically long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997), have become popular, often outperforming HMMs. Unlike HTM networks, neither HMMs nor LSTMs attempt to model biology in any detail; as such they provide no insights into neuronal or neocortical functions. The primary functional advantages of the HTM model over both these techniques are its ability to learn continuously, its superior robustness, and its ability to make multiple simultaneous predictions. A more detailed comparison can be found in S1 Table.

A number of papers have studied spiking neuron models (Ghosh-Dastidar and Adeli, 2009; Maass, 1997) in the context of sequences. These models are more biophysically detailed than the neuron models used in the machine learning literature. They show how spike-timing-dependent plasticity (STDP) can lead to a cell becoming responsive to a particular sequence of presynaptic spikes and to a specific time delay between each spike (Rao and Sejnowski, 2000; Ruf and Schmitt, 1997). These models are at a lower level of detail than the HTM model proposed in this paper. They explicitly model integration times of postsynaptic potentials and the corresponding time delays are typically sub-millisecond to a few milliseconds. They also typically deal with a very small subset of the synapses and do not explicitly model non-linear active dendrites. The focus of our work has been at a higher level. The work presented in this paper is a model of the full set of synapses and active dendrites on a neuron, of a networked layer of such neurons and the emergence of a computationally sophisticated sequence memory. An interesting direction for future research is to connect these two levels of modeling, i.e. to create biophysically detailed models that operate at the level of a complete layer of cells. Some progress is reported in (Billaudelle and Ahmad, 2015), but there remains much to do on this front.

A key consideration in learning algorithms is the issue of generalization, or the ability to robustly deal with novel patterns. The sequence memory mechanism we have outlined learns by forming synapses to small samples of active neurons in streams of sparse patterns. The properties of sparse representations naturally allow such a system to generalize. Two randomly selected sparse patterns will have very little overlap. Even a small overlap (such as 20%) is highly significant and implies that the representations share significant semantic meaning. Dendritic thresholds are lower than the actual number of synapses on each segment, thus segments will recognize novel but semantically related patterns as similar. The system will see similarity between different sequences and make novel predictions based on analogy.

Recently we showed that our sequence memory method can learn a predictive model of sensory-motor sequences (Cui et al., 2015). We also see it is likely that cortical motor sequences are generated using a variation of the same network model. Understanding how layers of cells can perform these different functions and how they work together is the focus of our current research.

5. Materials and Methods

Here we formally describe the activation and learning rules for an HTM sequence memory network. There are three basic aspects to the rules: initialization, computing cell states, and updating synapses on dendritic segments. These steps are described below, along with notation and some implementation details.

Notation: Let N represent the number of mini-columns in the layer, M the number of cells per column, and NM the total number of cells in the layer. Each cell can be in an active state, in a predictive (depolarized) state, or in a non-active state. Each cell maintains a set of segments each with a number of synapses. (In this figure we use the term "synapse" to refer to "potential synapses" as described in the body of the paper. Thus at any point in time some of the synapses will have a weight of 0 and some will have a weight of 1.) At any time step t, the set of active cells is represented by the M×N binary matrix $\mathbf{A}^t$, where a_{ij}^t is the activity of the i'th cell in the j'th column. Similarly, the M×N binary matrix $\mathbf{\Pi}^t$ denotes cells in a predictive state at time t, where π_{ij}^t is the predictive state of the i'th cell in the j'th column.

Each cell is associated with a set of distal segments, $\mathbf{D}_{ij}$, such that $\mathbf{D}_{ij}^d$ represents the d'th segment of the i'th cell in the j'th column. Each distal segment contains a number of synapses, representing lateral connections from a subset of the other $NM - 1$ cells. Each synapse has an associated permanence value (see Supplemental Fig. 2). Therefore, $\mathbf{D}_{ij}^d$ itself is also an M×N sparse matrix. If there are s potential synapses associated with the segment, the matrix contains s non-zero elements representing permanence values. A synapse is considered connected if its permanence value is above a connection threshold. We use $\tilde{\mathbf{D}}_{ij}^d$ to denote a binary matrix containing only the connected synapses.

1) Initialization: the network is initialized such that each segment contains a set of potential synapses (i.e. with non-zero permanence value) to a randomly chosen subset of cells in the layer. The permanence values of these potential synapses are chosen randomly: initially some are connected (above threshold) and some are unconnected.

2) Computing cell states: All the cells in a mini-column share the same feed forward receptive fields. We assume that an inhibitory process has already selected a set of k columns that best match the current feed forward input pattern. We denote this set as $\mathbf{W}^t$. The active state for each cell is calculated as follows:

$$a_{ij}^{t} = \begin{cases} 1 \; if j \in \boldsymbol{W}^{t} \; and \; \pi_{ij}^{t-1} = 1 \\ 1 \; if j \in \boldsymbol{W}^{t} \; and \sum_{i} \pi_{ij}^{t-1} = 0 \\ 0 \; otherwise \end{cases} \quad (2)$$

The first line will activate a cell in a winning column if it was previously in a predictive state. If none of the cells in a winning column were in a predictive state, the second line will activate all cells in that column. The predictive state for the current time step is then calculated as follows:

$$\pi_{ij}^{t} = \begin{cases} 1 \; if \exists_{d} \left\| \widetilde{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t} \right\|_{1} > \theta \\ 0 \; otherwise \end{cases} \quad (3)$$

Threshold θ represents the NMDA spiking threshold and ∘ represents element-wise multiplication. At a given point in time, if there are more than θ connected synapses with active presynaptic cells, then that segment will be active (generate an NMDA spike). A cell will be depolarized if at least one segment is active.

3) Updating segments and synapses: the HTM synaptic plasticity rule is a Hebbian-like rule. If a cell was correctly predicted (i.e. it was previously in a depolarized state and subsequently became active via feedforward input), we reinforce the dendritic segment that was active and caused the depolarization. Specifically, we choose those segments $\mathbf{D}_{\mathrm{ij}}^{\mathrm{d}}$ such that:

$$\forall_{j \in W^{t}} \left(\pi_{ij}^{t-1} > 0 \right) \text{ and } \left\| \widetilde{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t-1} \right\|_{1} > \theta \quad (4)$$

The first term selects winning columns that contained correct predictions. The second term selects those segments specifically responsible for the prediction.

If a winning column was unpredicted, we need to select one cell that will represent the context in the future if the current sequence transition repeats. To do this we select the cell with the segment that was closest to being active, i.e. the segment that had the most input even though it was below threshold. Let $\dot{\mathrm{D}}_{\mathrm{ij}}^{\mathrm{d}}$ denote a binary matrix containing only the positive entries in $\mathrm{D}_{\mathrm{ij}}^{\mathrm{d}}$. We reinforce a segment where the following is true:

$$\forall_{j \in W^{t}} \left(\sum_{i} \pi_{ij}^{t-1} = 0 \right) \text{ and } \left\| \dot{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t-1} \right\|_{1} = max_{i} \left(\left\| \dot{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t-1} \right\|_{1} \right) \quad (5)$$

Reinforcing the above segments is straightforward: we wish to reward synapses with active presynaptic cells and punish synapses with inactive cells. To do that we decrease all the permanence values by a small value p^{-} and increase the permanence values corresponding to active presynaptic cells by a larger value p^{+}:

$$\Delta \boldsymbol{D}_{ij}^{d} = p^{+} \left(\dot{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t-1} \right) - p^{-} \dot{\boldsymbol{D}}_{ij}^{d} \quad (6)$$

The above rules deal with cells that are currently active. We also apply a very small decay to active segments of cells that did not become active. This can happen if segments were mistakenly reinforced by chance:

$$\Delta \boldsymbol{D}_{ij}^{d} = p^{-} \dot{\boldsymbol{D}}_{ij}^{d} \text{ where } a_{ij}^{t} = 0 \; and \left\| \widetilde{\boldsymbol{D}}_{ij}^{d} \circ \boldsymbol{A}^{t-1} \right\|_{1} > \theta \quad (7)$$

The matrices $\Delta \mathbf{D}_{\mathrm{ij}}^{\mathrm{d}}$ are added to the current matrices of permanence values at every time step.

Implementation details: in our software implementation, we make some simplifying assumptions that greatly speed up simulation time for larger networks. Instead of explicitly initializing a complete set of synapses across every segment and every cell, we greedily create segments on a random cell and initialize potential synapses on that segment by sampling from currently active cells. This happens only when there is no match to any existing segment. In our simulations $\mathrm{N} = 2048, \mathrm{M} = 32, \mathrm{k} = 40$. We typically connect between 20 and 40 synapses on a segment, and θ is around 15. Permanence values vary from 0 to 1 with a connection threshold of 0.5. $\mathrm{p}^{+}$and p^{-} are small values that are tuned based on the individual dataset but typically less than 0.1. The full source code for the implementation is available on Github at https://github.com/numenta/nupic

6. REFERENCES

Ahmad, S., and Hawkins, J. (2015). Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory. arXiv:1503.07469 [q-bio.NC]

Antic, S. D., Zhou, W. L., Moore, A. R., Short, S. M., and Ikonomu, K. D. (2010). The decade of the dendritic NMDA spike. *J. Neurosci. Res.* 88, 2991–3001. doi:10.1002/jnr.22444.

Billaudelle, S., and Ahmad, S. (2015). Porting HTM Models to the Heidelberg Neuromorphic Computing Platform. arXiv:1505.02142 [q-bio.NC]

Chklovskii, D. B., Mel, B. W., and Svoboda, K. (2004). Cortical rewiring and information storage. *Nature* 431, 782–8. doi:10.1038/nature03012.

Cichon, J., and Gan, W.-B. (2015). Branch-specific dendritic Ca2+ spikes cause persistent synaptic plasticity. *Nature* 520, 180–5.

Cui, Y., Ahmad, S., Surpur, C., and Hawkins, J. (2015). Maintaining stable perception during active exploration. in *Cosyne Abstracts* (Salt Lake City).

Faisal, A. A., Selen, L. P. J., and Wolpert, D. M. (2008). Noise in the nervous system. *Nat. Rev. Neurosci.* 9, 292–303.

Ghosh-Dastidar, S., and Adeli, H. (2009). Spiking neural networks. *Int. J. Neural Syst.* 19, 295–308.

Golding, N. L., Jung, H. Y., Mickus, T., and Spruston, N. (1999). Dendritic calcium spike initiation and repolarization are controlled by distinct potassium channel subtypes in CA1 pyramidal neurons. *J. Neurosci.* 19, 8789–98.

Hawkins, J., Ahmad, S., and Dubinsky, D. (2011). Cortical learning algorithm and hierarchical temporal memory. http://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf

Hawkins, J., and Blakeslee, S. (2004). *On Intelligence*. New York: Henry Holt and Company.

Hochreiter, S., and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Comput.* 9, 1735–1780.

Holtmaat, A., and Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. *Nat. Rev. Neurosci.* 10, 647–58.

Hu, H., Gan, J., and Jonas, P. (2014). Fast-spiking, parvalbumin+ GABAergic interneurons: From cellular design to microcircuit function. *Science (80-.).* 345, 1255263–1255263.

Kerchner, G. A., and Nicoll, R. A. (2008). Silent synapses and the emergence of a postsynaptic mechanism for LTP. *Nat. Rev. Neurosci.* 9, 813–25.

Knott, G. W., Quairiaux, C., Genoud, C., and Welker, E. (2002). Formation of dendritic spines with GABAergic synapses induced by whisker stimulation in adult mice. *Neuron* 34, 265–73.

Lamme, V. A., Supèr, H., and Spekreijse, H. (1998). Feedforward, horizontal, and feedback processing in the visual cortex. *Curr. Opin. Neurobiol.* 8, 529–35.

Larkum, M. (2013). A cellular mechanism for cortical associations: an organizing principle for the cerebral cortex. *Trends Neurosci.* 36, 141–51. doi:10.1016/j.tins.2012.11.006.

Larkum, M. E., and Nevian, T. (2008). Synaptic clustering by dendritic signalling mechanisms. *Curr. Opin. Neurobiol.* 18, 321–31. doi:10.1016/j.conb.2008.08.013.

Larkum, M. E., Nevian, T., Sandler, M., Polsky, A., and Schiller, J. (2009). Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: a new unifying principle. *Science* 325, 756–760. doi:10.1126/science.1171958.

Larkum, M. E., Zhu, J. J., and Sakmann, B. (1999). A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature* 398, 338–41.

Lashley, K. (1951). “The problem of serial order in behavior,” in *Cerebral mechanisms in behavior*, ed. L. Jeffress (New York: Wiley), 112–131.

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

Losonczy, A., Makara, J. K., and Magee, J. C. (2008). Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature* 452, 436–41. doi:10.1038/nature06725.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks* 10, 1659–1671. doi:10.1016/S0893-6080(97)00011-7.

Major, G., Larkum, M. E., and Schiller, J. (2013). Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.* 36, 1–24. doi:10.1146/annurev-neuro-062111-150343.

Major, G., Polsky, A., Denk, W., Schiller, J., and Tank, D. W. (2008). Spatiotemporally graded NMDA spike/plateau potentials in basal dendrites of neocortical pyramidal neurons. *J. Neurophysiol.* 99, 2584–601.

Martin, K. A. C., and Schröder, S. (2013). Functional heterogeneity in neighboring neurons of cat primary visual cortex in response to both artificial and natural stimuli. *J. Neurosci.* 33, 7325–44.

Megías, M., Emri, Z., Freund, T. F., and Gulyás, A. I. (2001). Total number and distribution of inhibitory and excitatory synapses on hippocampal CA1 pyramidal cells. *Neuroscience* 102, 527–40.

Niell, C. M., Meyer, M. P., and Smith, S. J. (2004). In vivo imaging of synapse formation on a growing dendritic arbor. *Nat. Neurosci.* 7, 254–60.

Nikolić, D., Häusler, S., Singer, W., and Maass, W. (2009). Distributed fading memory for stimulus properties in the primary visual cortex. *PLoS Biol.* 7, e1000260.

Olshausen, B. A., and Field, D. J. (2004). Sparse coding of sensory inputs. *Curr. Opin. Neurobiol.* 14, 481–487. doi:10.1016/j.conb.2004.07.007.

Petreanu, L., Mao, T., Sternson, S. M., and Svoboda, K. (2009). The subcellular organization of neocortical excitatory connections. *Nature* 457, 1142–5.

Poirazi, P., Brannon, T., and Mel, B. W. (2003). Pyramidal neuron as two-layer neural network. *Neuron* 37, 989–99.

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

Rabiner, L., and Juang, B. (1986). An introduction to hidden Markov models. *IEEE ASSP Mag.* 3, 4–16.

Rah, J.-C., Bas, E., Colonell, J., Mishchenko, Y., Karsh, B., Fetter, R. D., et al. (2013). Thalamocortical input onto layer 5 pyramidal neurons measured using quantitative large-scale array tomography. *Front. Neural Circuits* 7, 177.

Rao, R. P. N., and Sejnowski, T. J. (2000). Predictive Sequence Learning in Recurrent Neocortical Circuits. in *in Advances in Neural Information Processing Systems 12*, 164–170. doi:10.1.1.45.9739.

Ruf, B., and Schmitt, M. (1997). Learning temporally encoded patterns in networks of spiking neurons. *Neural Process. Lett.*, 9–18. doi:10.1023/A:1009697008681.

Schiller, J., Major, G., Koester, H. J., and Schiller, Y. (2000). NMDA spikes in basal dendrites of cortical pyramidal neurons. *Nature* 404, 285–9.

Schiller, J., and Schiller, Y. (2001). NMDA receptor-mediated dendritic spikes and coincident signal amplification. *Curr. Opin. Neurobiol.* 11, 343–8.

Spruston, N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nat. Rev. Neurosci.* 9, 206–21.

Stuart, G. J., and Häusser, M. (2001). Dendritic coincidence detection of EPSPs and action potentials. *Nat. Neurosci.* 4, 63–71.

Trachtenberg, J. T., Chen, B. E., Knott, G. W., Feng, G., Sanes, J. R., Welker, E., et al. (2002). Long-term in vivo imaging of experience-dependent synaptic plasticity in adult cortex. *Nature* 420, 788–94.

Vinje, W. E., and Gallant, J. L. (2002). Natural Stimulation of the Nonclassical Receptive Field Increases Information Transmission Efficiency in V1. *J. Neurosci.* 22, 2904–2915.

Yen, S.-C., Baker, J., and Gray, C. M. (2007). Heterogeneity in the responses of adjacent neurons to natural stimuli in cat striate cortex. *J. Neurophysiol.* 97, 1326–1341. doi:10.1167/7.9.326.

Yoshimura, Y., Sato, H., Imamura, K., and Watanabe, Y. (2000). Properties of horizontal and vertical inputs to pyramidal cells in the superficial layers of the cat visual cortex. *J. Neurosci.* 20, 1931–40.

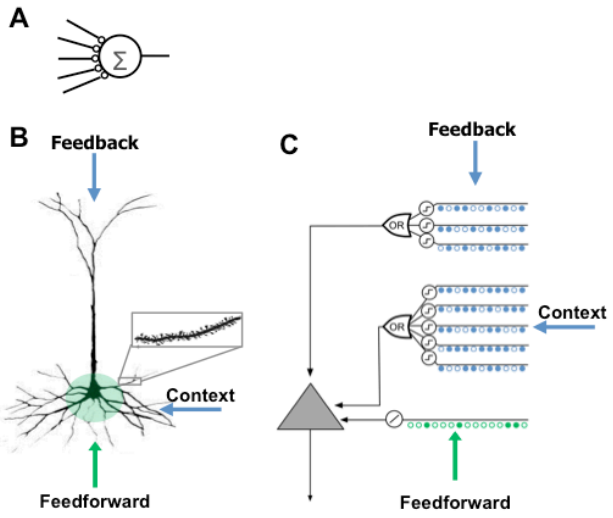


Figure 1: Comparison of neuron models. **A**) The neuron model used in most artificial neural networks has few synapses and no dendrites. **B**) A neocortical pyramidal neuron has thousands of excitatory synapses located on dendrites (inset). The co-activation of a set of synapses on a dendritic segment will cause an NMDA spike and depolarization at the soma. There are three sources of input to the cell. The feedforward inputs (shown in green) which form synapses proximal to the soma, directly lead to action potentials. NMDA spikes generated in the more distal basal and apical dendrites depolarize the soma but typically not sufficiently to generate a somatic action potential. **C**) An HTM model neuron models dendrites and NMDA spikes with an array of coincident detectors each with a set of synapses (only a few of each are shown).

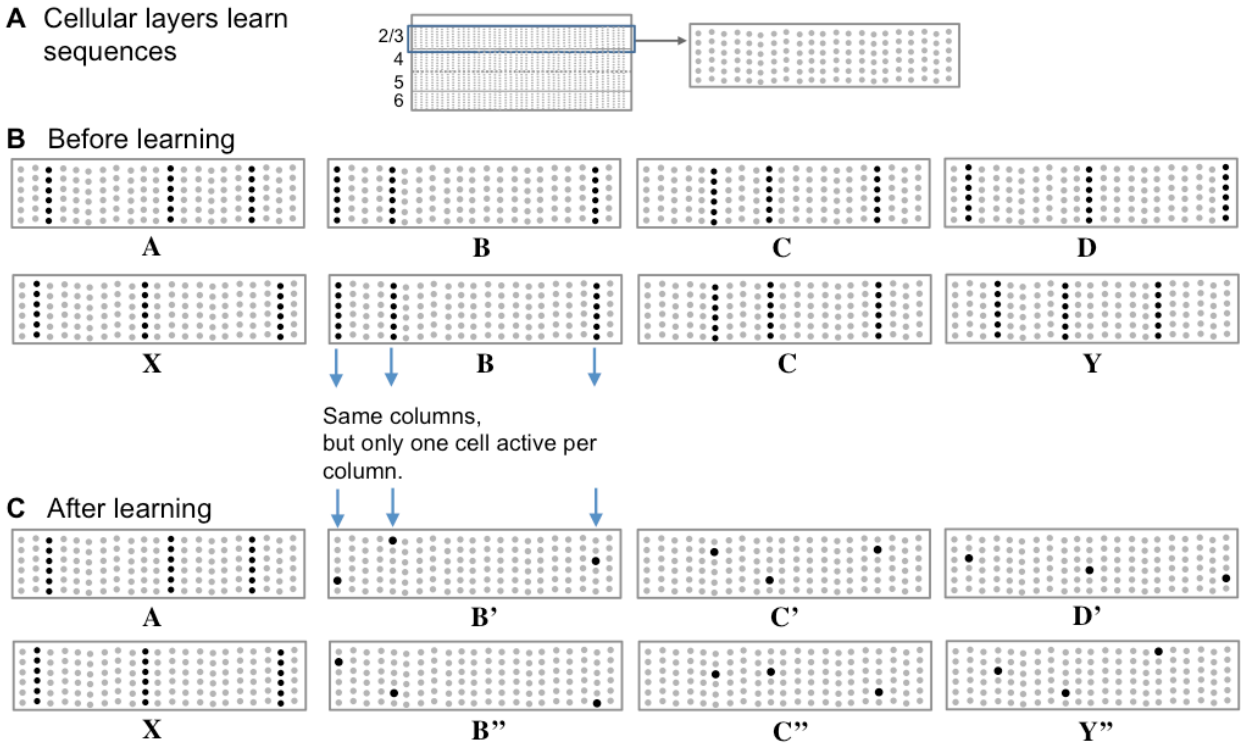


Figure 2: Representing sequences in cortical cellular layers. **A)** The neocortex is divided into cellular layers. The panels in this figure show part of one generic cellular layer. For clarity, the panels only show 21 mini-columns with 6 cells per column. **B)** Input sequences ABCD and XBCY are not yet learned. Each sequence element invokes a sparse set of mini-columns, only three in this illustration. All the cells in a mini-column become active if the input is unexpected, which is the case prior to learning the sequences. **C)** 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''. Because C' and C'' are unique, they can invoke the correct high-order prediction of either Y or D.

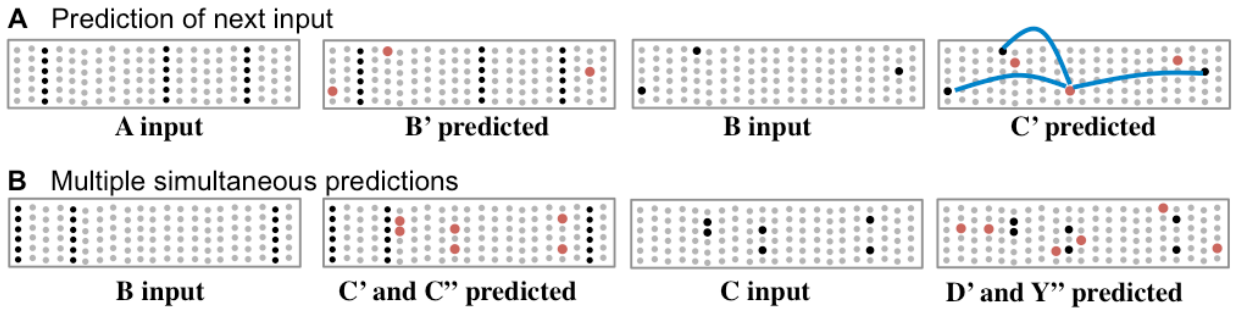


Figure 3: Basal connections to nearby neurons predict the next input. **A)** Using one of the sequences from Fig. 2, both active cells (black) and depolarized/predicted cells (red) are shown. The first panel shows the unexpected input A, which leads to a prediction of the next input B' (second panel). If the subsequent input matches the prediction then only the depolarized cells will become active (third panel), which leads to a new prediction (fourth panel). The lateral synaptic connections used by one of the predicted cells are shown in the rightmost panel. In a realistic network every predicted cell would have 15 or more connections to a subset of a large population of active cells. **B)** Ambiguous sub-sequence "BC" (which is part of both ABCD and XBCY) is presented to the network. The first panel shows the unexpected input B, which leads to a prediction of both C' and C''. The third panel shows the system after input C. Both sets of predicted cells become active, which leads to predicting both D and Y (fourth panel). In complex data streams there are typically many simultaneous predictions.

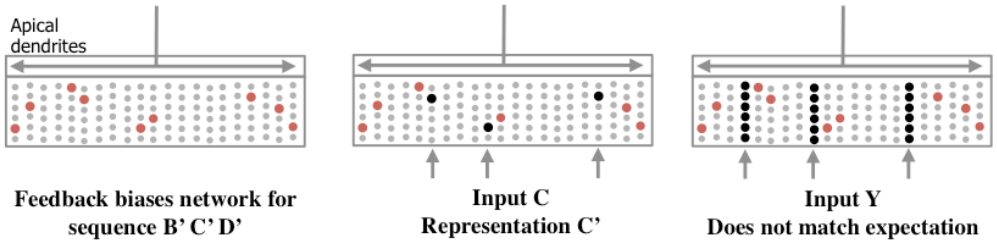


Figure 4: Feedback to apical dendrites predicts entire sequences.

This figure uses the same network and representations as Fig. 2. Area labeled "apical dendrites" is equivalent to layer 1 in neocortex; the apical dendrites (not shown) from all the cells terminate here. In the figure, the following assumptions have been made. The network has previously learned the sequence ABCD as was illustrated in Fig. 2. A constant feedback pattern was presented to the apical dendrites during the learned sequence, and the cells that participate in the sequence B'C'D' have formed synapses on their apical dendrites to recognize the constant feedback pattern.

After the feedback connections have been learned, presentation of the feedback pattern to the apical dendrites is simultaneously recognized by all the cells that would be active sequentially in the sequence. These cells, shown in red, become depolarized (left pane). When a new feedforward input arrives it will lead to the sparse representation relevant to the predicted sequence (middle panel). If a feedforward pattern cannot be interpreted as part of the expected sequence (right panel) then all cells in the selected columns become active indicative of an anomaly. In this manner apical feedback biases the network to interpret any input as part of an expected sequence and detects if an input does not match any one of the elements in the expected sequence.

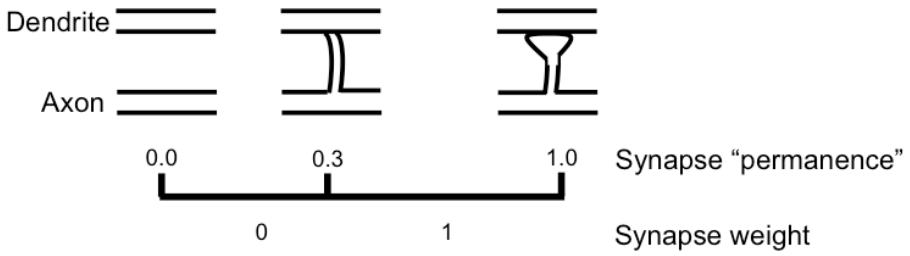


Figure 5: Learning by growing new synapses. Learning in an HTM neuron is modeled by the growth of new synapses from a set of potential synapses. A “permanence” value is assigned to each potential synapse and represents the growth of the synapse. Learning occurs by incrementing or decrementing permanence values. The synapse weight is a binary value set to 1 if the permanence is above a threshold.

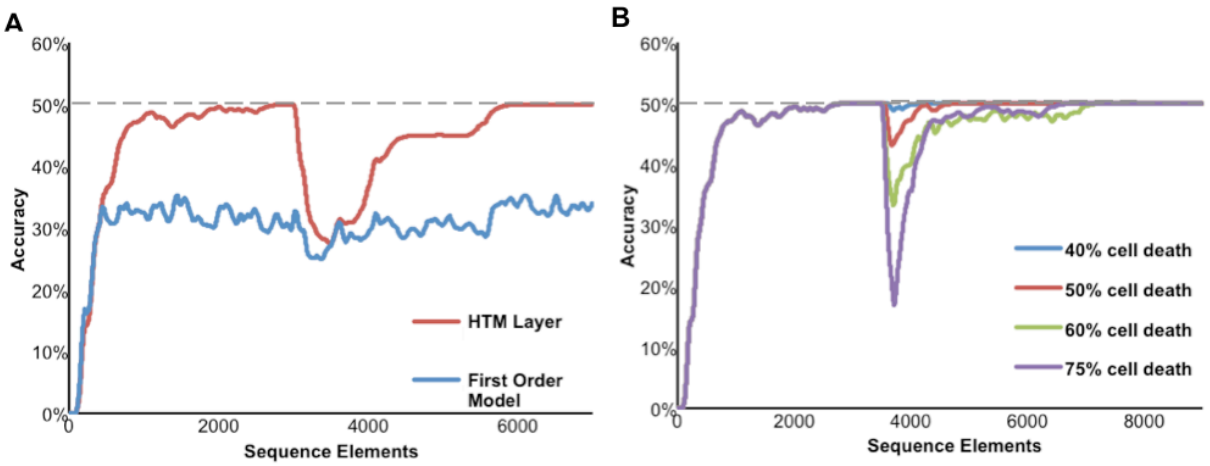


Figure 6: Simulation results of the sequence memory network. The input stream used for this figure contained high-order sequences mixed with random elements. The maximum possible average prediction accuracy of this data stream is 50%. **A)** High-order on-line learning. The red line shows the network learning and achieving maximum possible performance after about 2500 sequence elements. At element 3000 the sequences in the data stream were changed. Prediction accuracy drops and then recovers as the model learns the new temporal structure. For comparison, the lower performance of a first-order network is shown in blue. **B)** Robustness of the network to damage. After the network reached stable performance we inactivated a random selection of neurons. At up to 40% cell death there is almost no impact on performance. At greater than 40% cell death the performance of the network declines but then recovers as the network relearns using remaining neurons.

S1 Text. Chance of Error When Recognizing Large Patterns with a Few Synapses

Formula for calculating chance of error

A non-linear dendritic segment can robustly classify a pattern by sub-sampling (forming synapses to) a small number of cells from a large population. Assuming a random distribution of patterns, the exact probability of a false match, following is given by the following equation:

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

n = cell population size
a = number of active cells
s = number of synapses on segment
θ = NMDA spike threshold

Table A: Chance of error due to sub-sampling

This table demonstrates the effect of sub-sampling on the probability of a false match using the above equation. The chance of an error drops rapidly as the sampling size increases. A small number of synapses is sufficient for reliable matching.

s	Probability of false match
6	9.9×10^{-13}
8	9.8×10^{-17}
10	9.8×10^{-21}

$n = 200{,}000$
$a = 2{,}000$
$\theta = s$

Table B: Chance of error with addition of 50% noise immunity

This table demonstrates robustness to noise. By forming more synapses than required for an NMDA spike, a neuron can be robust to large amounts of noise and pattern variation and still have low probability of a false match. For example, with $s = 2\theta$ the system will be immune to 50% noise. The chance of an error drops rapidly as θ increases; even with noise a small number of synapses is sufficient for reliable matching.

θ	s	Probability of false match
6	12	8.7×10^{-10}
8	16	1.2×10^{-12}
10	20	1.6×10^{-15}
12	24	2.3×10^{-18}

$n = 200{,}000$
$a = 2{,}000$

Table C: Chance of error with addition of mixing synapses on a dendritic segment

This table demonstrates that mixing synapses for m different patterns on a single dendritic segment will still not cause unacceptable errors. By setting $s = 2m\theta$ we can see how a segment can recognize m independent patterns and still be robust to 50% noise. It is possible to get very high accuracy with larger m by using a slightly higher threshold.

θ	m	s	Probability of false match
10	2	40	6.3×10^{-12}
10	4	80	8.5×10^{-9}
10	6	120	4.2×10^{-7}
15	6	120	1.7×10^{-12}

$n = 200{,}000$
$a = 2{,}000$

	HTM	HMMs	LSTM
High order sequences	Yes	Limited	Yes
Discovers high order sequence structure	Yes	No	Yes
Local learning rules	Yes	No	No*
Continuous learning	Yes	No	No
Multiple simultaneous predictions	Yes	No	No
Unsupervised learning	Yes	Yes	No
Robustness and fault tolerance	Very high	No	Yes
Detailed mapping to neuroscience	Yes	No	No
Probabilistic model	No	Yes	No

S1 Table, Comparison of Common Sequence Memory Algorithms

Table comparing two common sequence memory algorithms (HMM and LSTM) to proposed model (HTM).
* Although weight updated rules are local, LSTMs require computing a global error signal that is then back propagated.

How Can We Be So Dense? The Benefits of Using Highly Sparse Representations

Subutai Ahmad [1] **Luiz Scheinkman** [1]
Numenta, Redwood City, California, USA

Abstract

Most artificial networks today rely on dense representations, whereas biological networks rely on sparse representations. In this paper we show how sparse representations can be more robust to noise and interference, as long as the underlying dimensionality is sufficiently high. A key intuition that we develop is that the ratio of the operable volume around a sparse vector divided by the volume of the representational space decreases exponentially with dimensionality. We then analyze computationally efficient sparse networks containing both sparse weights and activations. Simulations on MNIST and the Google Speech Command Dataset show that such networks demonstrate significantly improved robustness and stability compared to dense networks, while maintaining competitive accuracy. We discuss the potential benefits of sparsity on accuracy, noise robustness, hyperparameter tuning, learning speed, computational efficiency, and power requirements.

1. Introduction

The literature on sparse representations in neural networks dates back many decades, with neuroscience as one of the primary motivations. In 1988 Kanerva proposed the use of sparse distributed memories (Kanerva, 1988) to model the highly sparse representations seen in the brain. In 1997, (Olshausen & Field, 1997) showed that incorporating sparse priors and sparse cost functions in encoders can lead to receptive field representations that are remarkably close to what is observed in the primate visual cortex. More recently (Lee et al., 2008; Chen et al., 2018) showed hierarchical sparse representations that qualitatively lead to natural looking hierarchical feature detectors. (Lee et al., 2009; Nair & Hinton, 2009; Srivastava et al., 2013; Rawlinson et al., 2018) showed that introducing sparsity terms can sometimes lead to improved test set accuracies.

Despite the above literature the majority of neural networks today rely on dense representations. One exception is the pervasive use of dropout (Srivastava et al., 2014) as a regularizer. Dropout randomly "kills" a percentage of the units (in practice usually 50%) on every training input presentation. Variational dropout techniques tune the dropout rates individually per weight (Molchanov et al., 2017). Dropout introduces random sparse representations during learning, and has been shown to be an effective regularizer in many contexts.

In this paper we discuss certain inherent benefits of high dimensional sparse representations. We focus on robustness and sensitivity to interference. These are central issues with today's neural network systems where even small (Szegedy et al., 2013) and large (Rosenfeld et al., 2018) perturbations can cause dramatic changes to a network's output. We offer two main contributions. First, we analyze high dimensional sparse representations, and show that such representations are naturally more robust to noise and interference from random inputs. When matching sparse patterns, corrupted versions of a pattern are "close" to the original whereas random patterns are exponentially hard to match.

Our second contribution is an efficient construction of sparse deep networks that is designed to exploit the above properties. We implement networks where the weights for each unit in a layer randomly sample from a sparse subset of the source layer below. In addition the output of each layer is constrained such that only the k most active units are allowed to be non-zero, where k is much smaller than the number of units in that layer. In these networks, the number of non-zero products for each layer is approximately $(\text{sparsity of layer } i) \times (\text{sparse weights of layer } i+1)$. This formulation results in simple differentiable sparse layers that can be dropped into both standard linear and convolutional layers.

We demonstrate significantly improved robustness to noise for MNIST and the Google Speech Commands dataset, while maintaining competitive accuracy in the standard zero

[1] . Correspondence to: Subutai Ahmad, Luiz Scheinkman <[sahmad, lscheinkman]@numenta.com>.

noise scenario. We discuss the number of weights used by sparse networks in these datasets, and the impact of additional pruning. Our work extends the existing literature on sparse networks and pruning (see Section 5 for a comparison with some prior work). At the end of the paper we discuss some possible areas for future work.

2. High Dimensional Sparse Representations

In this section we develop some basic properties of sparse representations as they relate to noise robustness and interference. In a typical neural network an input vector is matched against a stored weight vector using a dot product. This is then followed by a threshold-like non-linearity such as $\tanh(\cdot)$ or $\text{ReLU}(\cdot)$.

Ideally we would like the outputs of each layer to be invariant to noise or corrupted inputs. When comparing two sparse vectors via a dot product, the results are unaffected by the zero components of either vector. A key quantity we consider is the ratio of the matching volume around a prototype vector divided by the volume of the whole space. The larger the match volume around a vector, the more robust it is to noise. The smaller the ratio, the less likely it is that random inputs can affect the match.

2.1. Matching Sparse Binary Vectors

We quantify the above ratio using binary vectors (following our previous work in (Ahmad & Hawkins, 2016)). In this section we show that the ratio decreases exponentially with increased dimensionality, while maintaining a large match volume. Let $\boldsymbol{x}$ be a binary vector of length n, and let $|\boldsymbol{x}|$ denote the number of non-zero entries. The dot product $\boldsymbol{x}_i \cdot \boldsymbol{x}_j$ counts the overlap, or number of shared bits, between two such vectors. We would like to understand the probability of two vectors having significant overlap, i.e. overlap greater than some threshold θ.

We define the overlap set, $\Omega^n(\boldsymbol{x}_i, b, k)$, as the set of all vectors of size k that have exactly b bits of overlap with $\boldsymbol{x_i}$. The number of such vectors can be calculated as:

$$|\Omega^n(\boldsymbol{x}_i, b, k)| = \binom{|\boldsymbol{x}_i|}{b}\binom{n - |\boldsymbol{x}_i|}{k - b} \tag{1}$$

The left half of the above product counts all the ways we can select exactly b bits out of active bits in $|\boldsymbol{x}_i|$. The right half counts the number of ways we can select the remaining $k - b$ bits from the components of $\boldsymbol{x}_i$ that are zero. The product of these two quantities represents the number of all vectors with exactly b bits of overlap with $|\boldsymbol{x}_i|$. We can now count the number of vectors that match $\boldsymbol{x}_i$, i.e. where $\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta$ as:

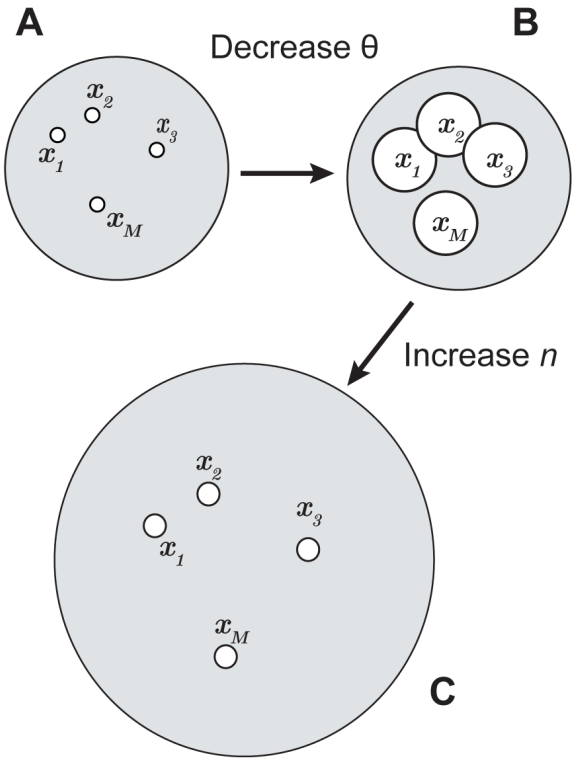


Figure 1. An illustration of the conceptual effect of decreasing the match threshold θ and increasing n, the dimensionality. The large grey circles denote the universe of possible patterns. The smaller circles each represent the set of matches around one vector. When θ is high (**A**), very few random vectors can match these vectors (small white circles). As you decrease θ, the set of potential matches increases (larger white circles in **B**). If you then increase n, the universe of possible patterns increases, and the relative sizes of the white circles shrink rapidly.

$$\sum_{b=\theta}^{|\boldsymbol{x}_i|} \mid \Omega^n(\boldsymbol{x}_i, b, |\boldsymbol{x}_j|) \mid \tag{2}$$

If we select vectors from a uniform random distribution, the probability of significant overlap can be calculated as:

$$P(\boldsymbol{x}_i \cdot \boldsymbol{x}_j \geq \theta) = \frac{\sum_{b=\theta}^{|\boldsymbol{x}_i|} \mid \Omega^n(\boldsymbol{x}_i, b, |\boldsymbol{x}_j|) \mid}{\binom{n}{|\boldsymbol{x}_j|}} \tag{3}$$

where $\binom{n}{|\boldsymbol{x}_j|}$ is the set of all possible comparison vectors.

2.2. Impact of Dimensionality and Sparsity

Two key factors in Eq. 3 are the number of non-zero components, $|\boldsymbol{x}_i|$, and the dimensionality, n. Figure 1 provides

an intuitive description of their impact. Assume we have M prototype vectors, and we want to match noisy versions of these vectors. Around each prototype there is a set of matching vectors. If the threshold is very high, the set of matching vectors is small (illustrated by the small circles in Figure 1A) and there will be quite a bit of space between these sets. As you decrease θ matching is less strict and you can match noisier versions of each prototype. The cost is that the chance of matching the other vectors also increases because there is less free space in between (Figure 1B). It turns out that for sparse vectors, this cost is offset as you increase n. That is, as n increases, the denominator in Eq. 3 (and the corresponding "free" space) increases much faster than the numerator. For a fixed sparsity level, you can maintain highly tolerant matches without the cost of additional false positives simply by increasing the dimensionality.

Fig 2 illustrates this trend for some example sparsities. In this figure we simulated matching with random vectors and plotted match rates with random vectors as a function of the number of active bits and the underlying dimensionality. In the simulation we repeatedly generated a random prototype vector with $|\boldsymbol{x}_i| = 24$ bits on and then attempted to match against random test vectors with a bits on. We matched using a threshold θ of 12 which meant that even vectors that were up to 50% different from $\boldsymbol{x}_i$ would match. We varied a and the dimensionality of the vectors, n.

The chart shows that for sparse binary vectors, match rates with random vectors drop rapidly as the underlying dimensionality increases. The horizontal line indicates the probability of matching $\boldsymbol{x}_i$ against dense vectors, with $a = n/2$. The probability of dense matches stays relatively high and unaffected by dimensionality, indicating that both sparseness and high dimensionality are key to robust matches. In (Ahmad & Hawkins, 2016) we develop additional properties, including the probability of false negatives.

2.3. Matching Sparse Scalar Vectors

Deep networks operate on scalar vectors, and in this section we consider how the above ideas apply to sparse scalar representations. Binary and scalar vectors are similar in that the components containing zero do not affect the dot product, and thus the combinatorics in Eq. 3 are still applicable. Eq. 1 represents the set of scalar vectors where the number of non-zero multiplies in the dot product is exactly b, and Eq. 3 represents the probability that the number of non-zero multiplies is $>= \theta$. However, an additional factor is the distribution of scalar values. If components in one vector are extremely large relative to θ, the likelihood of a significant match will be high even with a single shared non-zero component.

We wanted to see if the exponential drop in random matches for binary vectors, demonstrated by Figure 2, can be obtained using scalar vectors, and if so, the conditions under which they hold. Let $\boldsymbol{x}_w$ and $\boldsymbol{x}_i$ represent two sparse vectors such that $\|\boldsymbol{x}_w\|_0$ and $\|\boldsymbol{x}_i\|_0$ counts the number of non-zero entries in each. Let each non-zero component be independent and sampled from the distributions $P_{\theta_w}(x_w)$ and $P_{\theta_i}(x_i)$. The probability of a significant match is then:

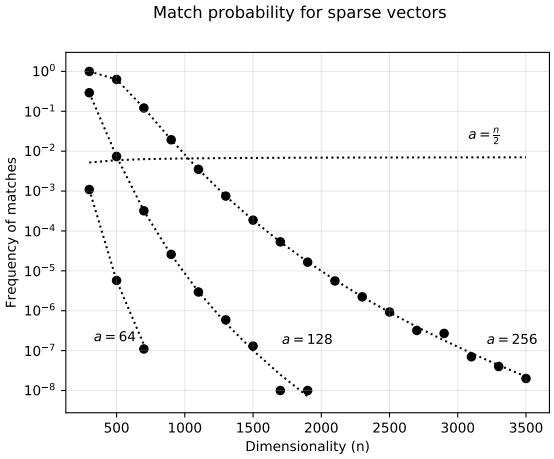


Figure 2. The probability of matches to random binary vectors (with a active bits) as a function of dimensionality, for various levels of sparsity. The probability decreases exponentially with n. Black circles denote the observed frequency of a match (based on a large number of trials). The dotted lines denote the theoretically predicted probabilities using Eq. 3.

$$P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta) = \frac{\sum_{b=\theta}^{\|\boldsymbol{x}_w\|_0} p_b \mid \Omega^n(\boldsymbol{x}_w, b, \|\boldsymbol{x}_i\|_0) \mid}{\binom{n}{\|\boldsymbol{x}_i\|_0}} \quad (4)$$

where p_b is the probability that the dot product is $>= \theta$ given that the overlap is exactly b components:

$$p_b = P(\boldsymbol{x}_w \cdot \boldsymbol{x}_i \geq \theta \mid \|\boldsymbol{x}_w \cdot \boldsymbol{x}_i\|_0 = b) \quad (5)$$

There does not appear to be a closed form way to compute p_b for normal or uniform distributions so we resort to simulations that mimic our network structure.

As before, we generated a large number of random vectors $\boldsymbol{x}_w$ and $\boldsymbol{x}_i$, and plotted the frequency of random matches. With $\|\boldsymbol{x}_w\|_0 = k$, we focus on simulations where the non-zero entries in $\boldsymbol{x}_w$ are uniform in $[-1/k, 1/k]$, and the non-zero entries in $\boldsymbol{x}_i$ are uniform in $S * [0, 2/k]$. We focus on this formulation because of the relationship to common network structures and weight initialization. $\boldsymbol{x}_w$ is a putative weight vector and $\boldsymbol{x}_i$ is an input vector to this layer from the

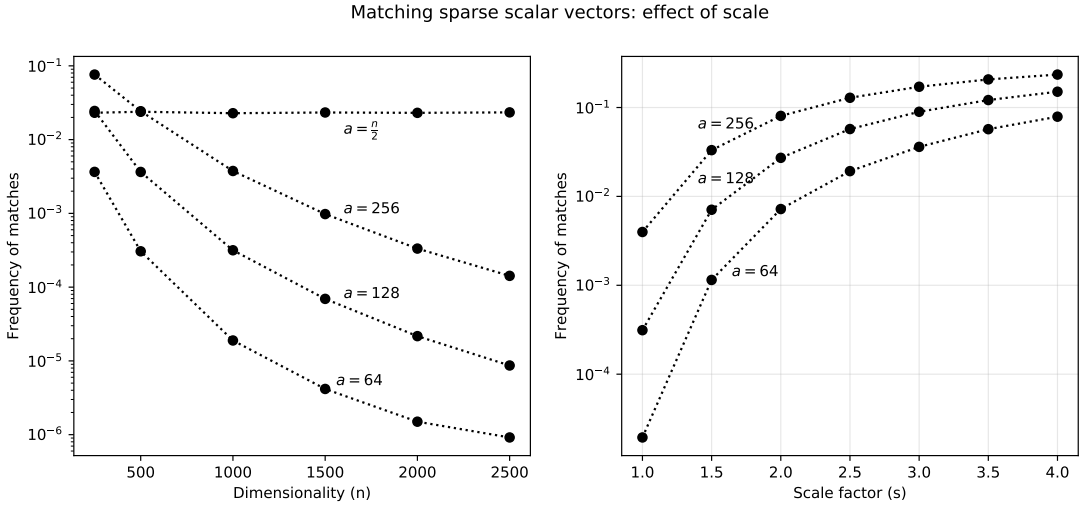


Figure 3. **Left**: The probability of matches to random scalar vectors (with a non-zero components) as a function of dimensionality, for various levels of sparsity. The probability of false matches decreases exponentially with n. Note that the probability for a dense vector, $a = \frac{n}{2}$ stays relatively high, and does not decrease with dimensionality. **Right**: The impact of scale on vector matches with a fixed $n = 1000$. The larger the scaling discrepancy, the higher the probability of a false match.

previous layer (we assume unit activations are positive, the result of a ReLU-like non-linearity). S controls the scale of $\boldsymbol{x}_i$ relative to $\boldsymbol{x}_w$.

Figure 3 (**left**) shows the behavior with $k = 32$ and $S = 1$. We varied the activity of the input vectors $\|\boldsymbol{x}_i\|_0 = a$ and the dimensionality of the vectors, n. We set $\theta = E[\boldsymbol{x}_w \cdot \boldsymbol{x}_w]/2.0$. The chart demonstrates that under these conditions we can achieve robust behavior similar to that of binary vectors. Figure 3 (**right**) plots the effect of S on the match probabilities with a fixed $n = 1000$. As this chart shows, the error increases significantly as S increases. Taken together, these results show that the fundamental robustness properties of binary sparse vectors can also hold for sparse scalar vectors, as long as the overall scaling of vectors are in a similar range.

2.4. Non-uniform Distribution of Vectors

Eq. 3 assumes the ideal case where vectors are chosen with a uniform random distribution. With a non-uniform distribution the error rates will be higher. The more non-uniform the distribution the worse the error rates. For example, if you mostly end up observing 10 inputs, your error rates will be bounded at around 10%. Thus, to optimize error rates, it is important to be as close to a uniform distribution as possible.

3. Sparse Network Description

Here we discuss a particular sparse network implementation that is designed to exploit Eq. 3. This implementation is an extension of our previous work on the HTM Spatial Pooler, a binary sparse coding algorithm that models sparse code generation in the neocortex (Hawkins et al., 2011; Cui et al., 2017). Specifically, we formulate a version of the Spatial Pooler that is designed to be a drop-in layer for neural networks trained with back-propagation. Our work is also closely related to previous literature on k-winner take all networks (Majani et al., 1989) and fixed sparsity networks (Makhzani & Frey, 2015).

Consider a network with L hidden layers. Let $\boldsymbol{y}^l$ denote the vector of outputs from layer l, respectively, with $\boldsymbol{y}^0$ as the input vector. $\boldsymbol{W}^l$ and $\boldsymbol{u}^l$ are the weights and biases for each layer. In a standard neural network the weights $\boldsymbol{W}^l$ are typically dense and initialized using a uniform random distribution. The feed forward outputs are then calculated as follows:

$$\hat{\boldsymbol{y}}^l = \boldsymbol{W}^l \cdot \boldsymbol{y}^{l-1} + \boldsymbol{u}^l$$
$$\boldsymbol{y}^l = f(\hat{\boldsymbol{y}}^l)$$

where f is any activation function, such as $\tanh(\cdot)$ or $\text{ReLU}(\cdot)$. (Figure 4 left.)

To implement our sparse networks, we make two modifications to this basic formulation (Figure 4 right.). First, we initialize the weights using a sparse random distribution, such

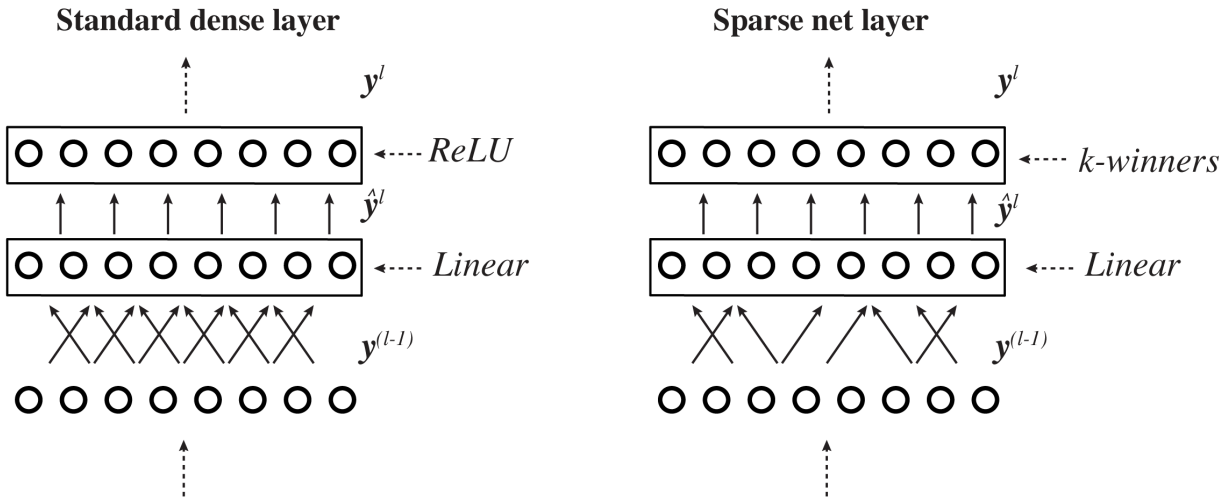


Figure 4. This figure illustrates the differences between a generic dense network layer (**left**) and a sparse network layer (**right**). In the sparse layer, the linear layer subsamples from its input layer (implemented via sparse weights, depicted with fewer arrows). In addition, the ReLU layer is replaced by a k-winners layer.

that only a fraction of the weights contain non-zero values. Non-zero weights are initialized using standard Kaiming initialization (He et al., 2015b). The rest of the connections are treated as non-existent, i.e. the corresponding weights are zero throughout the life of the network. Second, only the top-k active units within each layer are maintained in $\boldsymbol{y}^l$, and the rest set to zero. This k-winners step is non-linear and can be thought of as a substitute for the ReLU function. Instead of a threshold of 0, the threshold here is adaptive and corresponds to the k'th largest activation (Makhzani & Frey, 2013).

The layer can be trained using standard gradient descent. Similar to ReLU, the gradient of the layer is calculated as 1 above the threshold and 0 elsewhere. During inference we increase k by 50%, which led to slightly better accuracies. In all our simulations the last layer of each network is a standard linear output layer with log-softmax activation function.

3.1. Boosting

One practical issue with the above formulation is that it is possible for a small number of units to initially dominate and then, through learning, become active for a large percentage of patterns (this was also noted in (Makhzani & Frey, 2015; Cui et al., 2017)). Having a small number of active units negatively impacts the available representational volume. It is desirable for every unit to be equally active in order to maximize the robustness of the representation in Eq. 3. To address this we employ a boosting term (Hawkins et al., 2011; Cui et al., 2017) which favors units that have not been active recently. We compute a running average of each unit's duty cycle (i.e. how frequently it has been one of the top k units):

$$d_i^l(t) = (1-\alpha)d_i^l(t-1) + \alpha \cdot [i \in \text{topIndices}^l] \quad (6)$$

A boost coefficient b_i^l is then calculated for each unit based on the target duty cycle and the current average duty cycle:

$$b_i^l(t) = e^{\beta(\hat{a}^l - d_i^l(t))} \quad (7)$$

The target duty cycle $\hat{a}^l$ is a constant reflecting the percentage of units that are expected to be active, i.e. $\hat{a}^l = \frac{k}{|\boldsymbol{y}^l|}$. The boost factor, β, is a positive parameter that controls the strength of boosting. $\beta = 0$ implies no boosting ($b_i^l = 1$), and higher numbers lead to larger boost coefficients. In (Hawkins et al., 2011; Cui et al., 2017) we showed that Eq. 7 encourages each unit to have equal activation frequency and effectively maximizes the entropy of the layer.

The boost coefficients are used during the k-winners step to select which units remain active for this input. Through boosting, units which have not been active recently have a disproportionately higher impact and are more likely to win, whereas overly active units are de-emphasized. To determine the output of the layer, the non-boosted activity

Algorithm 1 k-winners layer

1: $\hat{\boldsymbol{y}}^l = \boldsymbol{w}^l \cdot \boldsymbol{y}^{(l-1)} + \boldsymbol{u}^l$
2: $b_i^l(t) = e^{\beta(\hat{a}^l - d_i^l(t))}$
3: $\text{topIndices}^l = topk(\boldsymbol{b}^l \odot \hat{\boldsymbol{y}}^l)$
4: $\boldsymbol{y}^l = 0$
5: $\boldsymbol{y}^l[\text{topIndices}^l] = \hat{\boldsymbol{y}}^l$
6: $d_i^l(t) = (1-\alpha)d_i^l(t-1) + \alpha \cdot [y_i^l(t) \in \text{topIndices}^l]$

of each winning unit is kept and the remaining units are set to zero. The duty cycle is then updated. The complete pseudo-code description for the k-winners layer is described in Algorithm 1. In our simulations we used $\beta = 1.0$ or 1.5 for all sparse simulations.

3.2. Sparse Convolutional Layers

We can apply the above algorithm to convolutional networks (CNNs) (LeCun et al., 1989). A canonical CNN layer uses a linear convolutional layer containing a number of filters, followed by a max-pooling (downsampling) layer, followed by ReLU. In order to implement sparse CNN layers, the k-winners layer is applied to the output of the max-pooling layer instead of ReLU (just as in our non-convolutional layers). However, since each filter in a CNN shares weights across the image, duty cycles are accumulated per filter. In our simulations dense and sparse CNN nets both have a hidden layer (which is dense or sparse, respectively) after the last convolutional layer, followed by a linear plus softmax layer. We used $5X5$ filters throughout with a stride of 1. In our tests, the weight sparsity of CNN layers did not impact the results. We suspect this is due to the small size of each kernel and did not use sparse weights for the CNN filters in our experiments.

4. Results

4.1. MNIST

We first trained our networks on MNIST (LeCun et al., 1998). We trained both dense and sparse implementations. Each network consisted of one or two convolutional layers, followed by a hidden layer, followed by a linear + softmax output layer. Sparse nets consisted of sparse convolutional layers followed by a sparse hidden layer.

Networks were trained using standard stochastic gradient descent to minimize cross entropy loss. We used starting learning rates in the range $0.01 - 0.04$, and the learning rate was decreased by a factor between 0.5 and 0.9 after each epoch. We also tried batch normalization (Ioffe & Szegedy, 2015) and found it did not help for MNIST (it did help significantly for Google Speech Commands results - see below). For sparse networks, we used a small mini-batch size (around 4), for the first epoch only, in order to let duty cycle calculations update frequently and settle. Hyperparameters such as the learning rate and network size were chosen using a validation set consisting of $10,000$ randomly chosen training samples. We then report final results on the test set using networks trained on the full training set.

NETWORK	TEST SCORE	NOISE SCORE
DENSE CNN-1	99.14 ± 0.03	$74{,}569 \pm 3{,}200$
DENSE CNN-2	99.31 ± 0.06	$97{,}040 \pm 2{,}853$
SPARSE CNN-1	98.41 ± 0.08	$100{,}306 \pm 1{,}735$
SPARSE CNN-2	99.09 ± 0.05	$103{,}764 \pm 1{,}125$
DENSE CNN-2 SP3	99.13 ± 0.07	$100{,}318 \pm 2{,}762$
SPARSE CNN-2 D3	98.89 ± 0.13	$102{,}328 \pm 1{,}720$
SPARSE CNN-2 W1	98.2 ± 0.19	$100{,}322 \pm 2{,}082$
SPARSE CNN-2 DSW	98.92 ± 0.09	$70{,}566 \pm 2{,}857$

Table 1. MNIST results for dense and sparse architectures. We show classification accuracies and total noise scores (the total number of correct classification for all noise levels). Results are averaged over 10 random seeds, $\pm$ one standard deviation. CNN-1 and CNN-2 indicate one or two convolutional layers, respectively.

Results Without Noise: State of the art accuracies on MNIST using convolutional neural networks (without distortions or other training augmentation) are in the range $98.3 - 99\%$ respectively[1]. Table 1 (left column) lists the classification accuracies for the networks in our experiments. Our accuracies are in the same range, for both sparse and dense networks. Table 3 lists the key parameters for each of the listed networks (see also the next section for a more in-depth discussion).

Results With Noise: In order to test noise robustness we generated MNIST images with varying levels of additive noise. For each test image we randomly set $\eta\%$ of the pixels to a constant value near white (the constant value was two standard deviations over the mean pixel intensity). Figure 5 (A) shows sample images for different noise levels. We generated 11 different noise levels with η ranging between 0 and 0.5 in increments of 0.05. We also computed an overall **noise score** which counted the total number of correct classifications across all noise levels.

The right column of Table 1 shows the noise scores for each of the architectures. Networks in the top section of the table (Dense CNN-1 and Dense CNN-2) are composed of standard dense convolutional and hidden layers. Networks in the middle section (Sparse CNN-1 and Sparse CNN-2) are composed of sparse convolutional and sparse hidden layers. Networks in the last section contain a mixture of dense and sparse layers. Overall the architectures with

[1]Source: `http://yann.lecun.com/exdb/mnist`

sparse layers performed significantly better on the noise score than the fully dense networks. Sparse CNN-2, the two layer completely sparse network, had the best noise score. The two fully dense networks performed substantially worse than the others on noise, even though their test accuracies were comparable. Figure 5 plots the accuracy of fully dense and sparse networks at different noise levels. Note that raw test score was not a predictor of noise robustness, suggesting that focusing on pure test set accuracy alone is not sufficient for gauging performance under adverse conditions.

Ablation studies: In order to judge the relative contributions of sparse layers we ran experiments where we replaced various sparse components with their dense counterparts, i.e. dense CNNs with sparse hidden layers, and vice versa. Dense CNN-2 SP3 contained two dense CNN layers followed by the sparse third layer from Sparse CNN-2. Sparse CNN-2 D3 contained the same CNN layers as Sparse CNN-2 followed by the dense third layer from Dense CNN-2. Sparse CNN-2 W1 was identical to Sparse CNN-2 except that the weight sparsity was 1 (i.e. fully dense weights). Sparse CNN-2 DSW contained a third layer with dense outputs, but with a weight sparsity of 0.3%.

The results of these networks are shown in the bottom third of Table 1. From a noise robustness perspective, most of the variants (except for Sparse CNN-2 DSW) performed well, better than the best pure dense network. This supports the idea that sparsity in many forms may be helpful with robustness. It is interesting to note that the standard deviation of the noise score in these variants was also higher than that of the pure sparse networks. Overall the results with mixed networks were encouraging, and suggest a clear benefit to introducing sparsity at any level.

Impact of Dropout: The above results did not use dropout (Srivastava et al., 2014), which is generally thought to improve robustness. We found that dropout did occasionally improve the robustness of dense networks, but any improvements were modest and the dropout percentage had to be tuned carefully. For sparse nets dropout consistently reduced accuracies. Even with the optimal dropout percentage, the noise scores of dense networks were significantly lower than sparse nets.

4.2. Google Speech Commands Dataset

In order to test sparse nets on a different domain, we applied them to the Google Speech Commands dataset (GSC). This audio dataset was made publicly available in 2017 (Warden, 2017) and consists of 65,000 one-second long utterances of 30 keywords spoken by thousands of individuals. The dataset contains predefined training, validation, and test sets.

Reference convolutional nets using ten of the keyword categories (plus artificial "silence" and "unknown" categories created during training augmentation) achieve accuracies in the range $91 - 92\%$ (Sainath & Parada, 2015; Tang & Lin, 2017). In (Tang & Lin, 2017) they demonstrated improved accuracies in the range of $95 - 96\%$ using residual networks (ResNets (He et al., 2015b;a)).

NETWORK	TEST SCORE	NOISE SCORE
DENSE CNN-2 (DR=0.0)	96.37± 0.37	8,730± 471
DENSE CNN-2 (DR=0.5)	95.69± 0.48	7,681± 368
SPARSE CNN-2	96.65± 0.21	11,233± 1013
SUPER-SPARSE CNN-2	96.57± 0.16	10,752± 942

Table 2. Classification on Google Speech Commands for a number of architectures. We show test and noise scores, averaged over 10 random seeds, ± one standard deviation. Dr corresponds to different dropout levels.

A Kaggle competition using GSC (also limited to 10 categories) took place between November 2017 and early 2018[2]. For our simulations we use the preprocessing code provided by one of the top-10 contestants (Tuguldur, 2018) who achieved around $97 - 97.5\%$ accuracies using variants of ResNet and VGG (Simonyan & Zisserman, 2014) architectures. Following this implementation, audio samples in our simulations are converted to 32-band Mel spectograms before being fed to the network. During training we augment the data by randomly adjusting the amplitude, speed, and pitch of each training sample, and by randomly shifting and stretching samples in the frequency domain. No data augmentation is performed on the validation or test sets.

We trained dense and sparse convolutional networks, with hyperparameters chosen based on the validation set. We were able to achieve reasonable accuracies using two convolutional layers, followed by a hidden layer and then a linear + softmax output layer. Our sparse networks had sparse convolutional layers as well as a sparse hidden layer. Unlike MNIST we found that batch normalization (Ioffe & Szegedy, 2015) accelerated learning significantly, and we used it for every layer.

Using the above setup we were able to achieve test set accuracies in the range of $96.5 - 97.2\%$ classifying the ten categories corresponding to the digits "zero" through "nine". Table 2 (left column) shows mean accuracy on the test set. Both dense and sparse networks had about the same accuracy. Dropout had a negative effect on the accuracy. Table 3 lists the key parameters in each network.

Results With Noise: As with MNIST, we again created noisy versions of the test set. For each test audio sample $\boldsymbol{A}$ we generated a random white noise sample and blended

[2] https://www.kaggle.com/c/tensorflow-speech-recognition-challenge

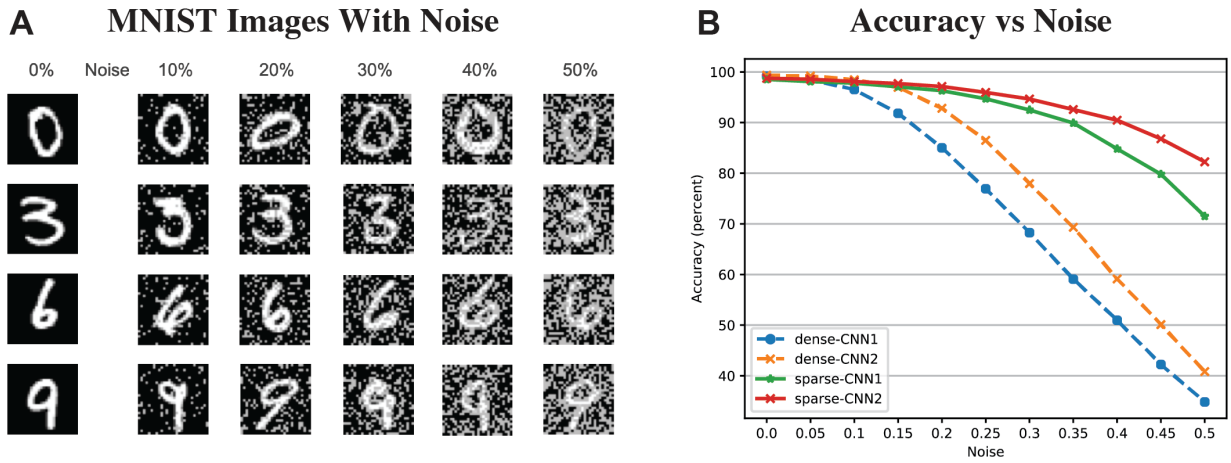


Figure 5. **A**. Example MNIST images with varying levels of noise. **B**. Classification accuracy as a function of noise level.

them together:

$$\boldsymbol{A}^* = (1-\eta)\boldsymbol{A} + \eta \text{whiteNoise}$$

We generated 11 different noise levels, with η ranging from 0 to 0.5 in increments of 0.05. Our overall noise score counted the total number of classifications across all noise levels.

As can be seen in Table 2 sparse networks performed significantly better than the best dense network. We included a "Super-Sparse CNN-2" with a significantly sparser hidden layer. The hidden layer for this network had 10% weight sparsity, and a lower output sparsity (Table 3). This network had slightly lower noise score, but its score was still significantly higher than that of the dense networks. Overall these results demonstrate that the robustness of sparse networks seen with MNIST can scale to other domains.

4.3. Computational Considerations

In standard networks, the size of each weight matrix is $|\boldsymbol{W}^l| = |\boldsymbol{y}^{l-1}||\boldsymbol{y}^l|$ and the order of complexity of the feed-forward operation can be approximated by the number of multiplications, $|\boldsymbol{y}^{l-1}|^2|\boldsymbol{y}^l|$. The computational efficiency of sparse systems is closely related to the fraction of non-zeros. In our sparse hidden layers, both activations and weight values are sparse and the number of non-zero product terms in the forward computation is proportional to $k^{l-1}w^l|\boldsymbol{y}^{l-1}||\boldsymbol{y}^l|$, where $0 < w^l \leq 1$ is the fraction of non-zero weights. In our convolutional layers, only activations values are sparse and the number of non-zero product terms in the forward computation is proportional to $k^{l-1} * \boldsymbol{K^l} * \boldsymbol{K^l} * |\boldsymbol{y}^l|$, where $\boldsymbol{K^l}$ is the kernel width of each filter.

As an example, the number of non-zero multiplies between the first two convolutional layers in the GSC Sparse CNN-2 network is $12,544 * 1600 * 6400 = 1.23X10^{10}$, about 10.5X smaller than the corresponding dense network. The number of non-zero multiplies between the second convolutional layer and the hidden layer in the same network is $200 * 640,000 * 1000 = 1.28X10^{11}$, about 20X smaller than the dense network. For Super Sparse CNN-2, that ratio is 35X as compared to the dense version.

As can be seen, the number of non-zeros products is significantly smaller in the sparse net implementations. Unfortunately we found that current versions of deep learning frameworks, including PyTorch and Tensorflow do not have adequate support for sparse matrices to exploit these properties, and our implementations ran at the same speed as the corresponding dense networks. We suspect this is due to the fact that highly sparse networks are not sufficiently popular in practice. We hope that studies such as this one will encourage highly optimized sparse implementations. (Note that such optimizations may be non-trivial as the set of k-winners changes on every step.) When this becomes feasible our numbers suggest there is a strong possibility for large performance gains and/or improvements in power usage. It is also worth noting that this reduction in computational complexity does not come at a cost. Rather, our experiments showed that sparse representations can lead to improved accuracies under noisy conditions.

5. Discussion

In this paper we illustrated benefits of sparse representations. We developed intuitions and theory for the structure of vec-

NETWORK	L1 F	L1 SPARSITY	L2 F	L2 SPARSITY	L3 N	L3 SPARSITY	WT SPARSITY
MNIST							
DENSE CNN-1	30	100%			1000	100%	100%
DENSE CNN-2	30	100%	30	100%	1000	100%	100%
SPARSE CNN-1	30	9.3%			150	33.3%	30%
SPARSE CNN-2	32	8.7%	64	29.3 %	700	14.3%	30%
DENSE CNN-2 SP3	30	100%	30	100%	700	14.3%	30%
SPARSE CNN-2 D3	32	8.7%	64	29.3 %	1000	100%	100%
SPARSE CNN-2 W1	32	8.7%	64	29.3 %	700	14.3%	100%
SPARSE CNN-2 DSW	32	8.7%	64	29.3 %	1000	100%	30%
GSC							
DENSE CNN-2	64	100%	64	100%	1000	100%	100%
SPARSE CNN-2	64	9.5%	64	12.5%	1000	10%	40%
SUPER SPARSE CNN-2	64	9.5%	64	12.5%	1500	6.7%	10%

Table 3. Key parameters for each network. L1F and L2F denote the number of filters at the corresponding CNN layer. L1,2,3 sparsity indicates k/n, the percentage of outputs that were enforced to be non-zero. 100% indicates a special case where we defaulted to traditional ReLU activations. Wt sparsity indicates the percentage of weights that were non-zero. All parameters are available in the source code.

tor matching in the context of binary sparse representations. We then constructed efficient neural network formulations of sparse networks that place internal representations in the sweet spot suggested by the theory. In particular we aim to match sparse activations with sparse weights in relatively high dimensional settings. A boosting rule was used to increase the overall entropy of the internal layers in order to maximize the utilization of the representational space. We showed that this formulation increases the overall robustness of the system to noisy inputs using MNIST and the Google Speech Command Dataset. Both dense and sparse networks showed high accuracies, but the sparse nets were significantly more robust. These results suggest that it is important to look beyond pure test set performance as test accuracy by itself is not a reliable indicator of overall robustness.

Our work extends the existing literature on sparsity and pruning. A very recent theoretical paper showed that simple linear sparse networks may be more robust to adversarial attacks (Guo et al., 2018). A number of papers have shown that it is possible to effectively introduce sparsity through pruning and retraining (Han et al., 2015; Frankle & Carbin, 2018; Lee et al., 2018). The mechanisms introduced here can be seen as complementary to those techniques. Our network enforces sparse weights from the beginning by construction, and sparse weights are learned as part of the training process. In addition, we reduce the overall computational complexity by enforcing sparse activations, which in turn significantly reduces the number of overall non-zero products. This should produce significant power savings for optimized hardware implementations.

We demonstrated increased robustness in our networks whereas the papers on pruning typically do not explicitly test robustness. It is possible that such networks are also more robust, though this remains to be tested. Pruning techniques in general are quite orthogonal to ours, and it may be feasible to combine them with the mechanisms discussed here.

In our work we did not attempt to introduce sparsity into the convolutional filters themselves. (Li et al., 2016) have shown it is sometimes possible to remove entire filters from large CNNs suggesting that sparsifying filter weights may also be possible, particularly in networks with larger filters. Introducing sparse convolutions within the context of the techniques in this paper is an area of future exploration. The techniques described here are straightforward to implement and can be extended to other architectures including RNNs. This is yet another promising area for future research.

5.1. Software

All code and experiments are available at https://github.com/numenta/htmpapers as open source.

Acknowledgements

We thank Jeff Hawkins, Ali Rahimi, and John Berkowitz for helpful discussions and comments.

References

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

Chen, Y., Paiton, D., & Olshausen, B. (2018). The Sparse Manifold Transform. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 31*, (pp. 10533–10544). Curran Associates, Inc.

Cui, Y., Ahmad, S., & Hawkins, J. (2017). The HTM Spatial Pooler a neocortical algorithm for online sparse distributed coding. *Frontiers in Computational Neuroscience*, *11*, 111.
URL https://www.frontiersin.org/articles/10.3389/fncom.2017.00111/abstract

Frankle, J., & Carbin, M. (2018). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks.
URL http://arxiv.org/abs/1803.03635

Guo, Y., Zhang, C., Zhang, C., & Chen, Y. (2018). Sparse DNNs with Improved Adversarial Robustness. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 31*, (pp. 240–249). Curran Associates, Inc.

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both Weights and Connections for Efficient Neural Network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 28*, (pp. 1135–1143). Curran Associates, Inc.

Hawkins, J., Ahmad, S., & Dubinsky, D. (2011). Cortical Learning Algorithm and Hierarchical Temporal Memory.
URL http://numenta.org/resources/HTM{_}CorticalLearningAlgorithms.pdf

He, K., Zhang, X., Ren, S., & Sun, J. (2015a). Deep Residual Learning for Image Recognition.
URL http://arxiv.org/abs/1512.03385

He, K., Zhang, X., Ren, S., & Sun, J. (2015b). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.
URL http://arxiv.org/abs/1502.01852

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.
URL http://arxiv.org/abs/1502.03167

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge, MA: The MIT Press.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*.

Lee, H., Ekanadham, C., & Ng, A. Y. (2008). Sparse deep belief net model for visual area V2. *Advances In Neural Information Processing Systems*.

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, (pp. 1–8).

Lee, N., Ajanthan, T., & Torr, P. H. S. (2018). SNIP: Single-shot Network Pruning based on Connection Sensitivity.
URL http://arxiv.org/abs/1810.02340

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). Pruning Filters for Efficient ConvNets.
URL http://arxiv.org/abs/1608.08710

Majani, E., Erlanson, R., & Abu-Mostafa, Y. S. (1989). On the k-winners-take-all network. In *Advances in neural information processing systems*, (pp. 634–642).

Makhzani, A., & Frey, B. (2013). k-Sparse Autoencoders.
URL http://arxiv.org/abs/1312.5663

Makhzani, A., & Frey, B. (2015). Winner-take-all autoencoders. *Advances in Neural Information Processing*.
URL http://papers.nips.cc/paper/5783-winner-take-all-autoencoders

Molchanov, D., Ashukha, A., & Vetrov, D. (2017). Variational Dropout Sparsifies Deep Neural Networks.
URL http://arxiv.org/abs/1701.05369

Nair, V., & Hinton, G. E. (2009). 3D Object Recognition with Deep Belief Nets. In Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, & A. Culotta (Eds.) *Advances in Neural Information Processing Systems 22*, (pp. 1339–1347). Curran Associates, Inc.

Olshausen, B. A., & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, *37*, 3311–3325.

Rawlinson, D., Ahmed, A., & Kowadlo, G. (2018). Sparse Unsupervised Capsules Generalize Better.
URL http://arxiv.org/abs/1804.06094

Rosenfeld, A., Zemel, R., & Tsotsos, J. K. (2018). The Elephant in the Room.
URL http://arxiv.org/abs/1808.03305

Sainath, T. N., & Parada, C. (2015). Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. URL http://arxiv.org/abs/1409.1556

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*, 1929–1958. URL http://jmlr.org/papers/v15/srivastava14a.html

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., & Schmidhuber, J. (2013). Compete to Compute. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.) *Advances in Neural Information Processing Systems 26*, (pp. 2310–2318). Curran Associates, Inc. URL http://papers.nips.cc/paper/5059-compete-to-compute.pdf

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. URL http://arxiv.org/abs/1312.6199

Tang, R., & Lin, J. (2017). Deep Residual Learning for Small-Footprint Keyword Spotting. URL https://arxiv.org/abs/1710.10361

Tuguldur, E.-O. (2018). pytorch-speech-commands. URL https://github.com/tugstugi/pytorch-speech-commands

Warden, P. (2017). Speech Commands: A public dataset for single-word speech recognition. *Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz.*

frontiers in Neural Circuits

HYPOTHESIS AND THEORY
published: 25 October 2017
doi: 10.3389/fncir.2017.00081

A Theory of How Columns in the Neocortex Enable Learning the Structure of the World

Jeff Hawkins, Subutai Ahmad and Yuwei Cui*

Numenta, Inc., Redwood City, CA, United States

Neocortical regions are organized into columns and layers. Connections between layers run mostly perpendicular to the surface suggesting a columnar functional organization. Some layers have long-range excitatory lateral connections suggesting interactions between columns. Similar patterns of connectivity exist in all regions but their exact role remain a mystery. In this paper, we propose a network model composed of columns and layers that performs robust object learning and recognition. Each column integrates its changing input over time to learn complete predictive models of observed objects. Excitatory lateral connections across columns allow the network to more rapidly infer objects based on the partial knowledge of adjacent columns. Because columns integrate input over time and space, the network learns models of complex objects that extend well beyond the receptive field of individual cells. Our network model introduces a new feature to cortical columns. We propose that a representation of location relative to the object being sensed is calculated within the sub-granular layers of each column. The location signal is provided as an input to the network, where it is combined with sensory data. Our model contains two layers and one or more columns. Simulations show that using Hebbian-like learning rules small single-column networks can learn to recognize hundreds of objects, with each object containing tens of features. Multi-column networks recognize objects with significantly fewer movements of the sensory receptors. Given the ubiquity of columnar and laminar connectivity patterns throughout the neocortex, we propose that columns and regions have more powerful recognition and modeling capabilities than previously assumed.

Keywords: neocortex, cortical columns, cortical layers, sensorimotor learning, hierarchical temporal memory

Edited by:
Fritjof Helmchen,
University of Zurich, Switzerland

Reviewed by:
Jason N. MacLean,
University of Chicago, United States
Heiko J. Luhmann,
Johannes Gutenberg-Universität Mainz, Germany
Rui Ponte Costa,
University of Bern, Switzerland

****Correspondence:***
Jeff Hawkins
jhawkins@numenta.com

Received: *13 July 2017*
Accepted: *10 October 2017*
Published: *25 October 2017*

Citation:
Hawkins J, Ahmad S and Cui Y (2017) A Theory of How Columns in the Neocortex Enable Learning the Structure of the World. Front. Neural Circuits 11:81. doi: 10.3389/fncir.2017.00081

INTRODUCTION

The neocortex is complex. Within its 2.5 mm thickness are dozens of cell types, numerous layers, and intricate connectivity patterns. The connections between cells suggest a columnar flow of information across layers as well as a laminar flow within some layers. Fortunately, this complex circuitry is remarkably preserved in all regions, suggesting that a canonical circuit consisting of columns and layers underlies everything the neocortex does. Understanding the function of the canonical circuit is a key goal of neuroscience.

Over the past century, several theories have been proposed to explain the existence of cortical layers and columns. One theory suggested these anatomical constructs minimize the amount of wiring in cortical tissues (Shipp, 2007). Some researchers suggested there should be functional

differentiation of different cortical layers that match the anatomical structure (Douglas and Martin, 2004). Others have proposed that long-range laminar connections contribute to attention-related changes in receptive field properties (Raizada and Grossberg, 2003). Recent advances in recording technologies now enable detailed recording of activity in the micro-circuitry of cortical columns. However, despite these advances, the function of networks of neurons organized in layers and columns remains unclear, and assigning any function to columns remains controversial (Horton and Adams, 2005).

Lacking a theory of why the neocortex is organized in columns and layers, almost all artificial neural networks, such as, those used in deep learning (LeCun et al., 2015) and spiking neural networks (Maass, 1997), do not include these features, introducing the possibility they may be missing key functional aspects of biological neural tissue. To build systems that work on the same principles as the neocortex we need an understanding of the functional role of columnar and laminar projections.

Cellular layers vary in the connections they make, but a few general rules have been observed. Cells in layers that receive direct feedforward input do not send their axons outside the local region and they do not form long distance horizontal connections within their own layer. Cells in layers that are driven by input layers form long range excitatory connections within their layer, and also send an axonal branch outside of the region, constituting an output of the region. This two-layer input-output circuit is a persistent feature of cortical regions. The most commonly recognized instance involves layer 4 and Layer 2/3. Layer 4 receives feedforward input. It projects to layer 2/3 which is an output layer (Douglas and Martin, 2004; Shipp, 2007). Upper layer 6 also receives feedforward input (Thomson, 2010). It projects to layer 5, which is an output layer (Douglas and Martin, 2004; Guillery and Sherman, 2011), and therefore layers 6 and 5 may be a second instance of the two-layer input-output circuit. The prevalence of this two-layer connection motif suggests it plays an essential role in cortical processing.

In this paper, we introduce a theory of how columns and layers learn the structure of objects in the world. It is a sensorimotor theory in that learning and inference require movement of sensors relative to objects. We also introduce a network model based on the theory. The network consists of one or more columns, where each column contains an input layer and an output layer. First, we show how even a single column can learn the structure of complex objects. A single column can only sense a part of an object at any point in time, however, the column will be exposed to multiple parts of an object as the corresponding sensory organ moves. While the activation in the input layer changes with each movement of the sensor, the activation in the output layer remains stable, associating a single output representation with a set of feature representations in the input layer. Thus, a single cortical column can learn models of complete objects through movement. These objects can be far larger than any individual cell's receptive field.

Next, we show how multiple columns collaborate via long-range intralaminar connections. At any point in time, each column has only partial knowledge of the object it is observing, yet adjacent columns are typically sensing the same object, albeit at different locations on the object. Long range excitatory connections in the output layer allow multiple columns to rapidly reach a consensus of what object is being observed. Although learning always requires multiple sensations via movement, inference with multiple columns can often occur in a single or just a few sensations. Through simulation we illustrate that our model can learn the structure of complex objects, it infers quickly, and it has high capacity.

A key component of our theory is the presence in each column of a signal representing location. The location signal represents an "allocentric" location, meaning it is a location relative to the object being sensed. In our theory, the input layer receives both a sensory signal and the location signal. Thus, the input layer knows both what feature it is sensing and where the sensory feature is on the object being sensed. The output layer learns complete models of objects as a set of features at locations. This is analogous to how computer-aided-design programs represent multi-dimensional objects.

Because different parts of a sensory array (for example different fingers or different parts of the retina) sense different parts of an object, the location signal must be calculated uniquely for each sensory patch and corresponding area of neocortex. We propose that the location signal is calculated in the sub-granular layers of cortical columns and is passed to input layer 4 via projections from layer 6.

It is important to note that we deduced the existence of the allocentric location signal. We first deduced its presence by considering how fingers can predict what they will sense while moving and touching an object. However, we believe the location signal is present in all neocortical regions. We show empirical evidence in support of this hypothesis. Although we cannot yet propose a complete mechanism for how the location signal is derived, the task of determining location and predicting new locations based on movement is similar to what grid cells do in the medial entorhinal cortex. Grid cells offer an existence proof that predictive models of allocentric location are possible, and they suggest mechanisms for how the location signal might be derived in cortical columns.

The theory is consistent with a large body of anatomical and physiological evidence. We discuss this support and propose several predictions that can be used to further test the theory.

MODEL

Motivation

Our research is focused on how networks of neurons in the neocortex learn predictive models of the world. Previously, we introduced a network (Hawkins and Ahmad, 2016) that learns a predictive model of naturally changing sensory sequences. In the present paper, we extend this network to address the related question of how the neocortex learns a predictive model of static objects, where the sensory input changes due to our own movement.

A simple thought experiment may be useful to understand our model. Imagine you reach your hand into a black box and try to determine what object is in the box, say a coffee cup. Using only one finger it is unlikely you could identify the object with a single

touch. However, after making one contact with the cup, you move your finger and touch another location, and then another. After a few touches, you identify the object as a coffee cup. Recognizing the cup requires more than just the tactile sensation from the finger, the brain must also integrate knowledge of how the finger is moving, and hence where it is relative to the cup. Once you recognize the cup, each additional movement of the finger generates a prediction of where the finger will be on the cup after the movement, and what the finger will feel when it arrives at the new location. This is the first problem we wanted to address, how a small sensory array (e.g., the tip of a finger) can learn a predictive model of three dimensional objects by integrating sensation and movement-derived location information.

If you use two fingers at a time you can identify the cup with fewer movements. If you use five fingers you will often be able to identify an object with a single grasp. This is the second problem we wanted to address, how a set of sensory arrays (e.g., tips of multiple fingers) work together to recognize an object faster than they can individually.

Somatic inference is obviously a sensorimotor problem. However, vision and audition are also sensorimotor tasks. Therefore, the mechanisms underlying sensorimotor learning and inference should exist in all sensory regions, and any proposed network model should map to the detailed anatomical and physiological properties that exist in all cortical regions. This mapping, an explanation of common cortical circuitry, is a third goal of our model.

Model Description

Our model extends previous work showing how a single layer of pyramidal neurons can learn sequences and make predictions (Hawkins and Ahmad, 2016). The current model consists of two layers of pyramidal neurons arranged in a column. The model has one or more of these columns (**Figure 1A**). Each cortical column processes a subset of the sensory input space and is exposed to different parts of the world as the sensors move. The goal is to have the output layer of each column converge on an object representation that is consistent with the accumulated sensations over time and across all columns.

The input layer of each column in our model receives a sensory input and a location input. The sensory input is a sparse binary array representing the current feature in its input space. The location input is a sparse binary array representing the location of the feature on the object. There are numerous observations in the neocortex that receptive fields are modified by location information. Grid cells in the entorhinal cortex also solve a similar location encoding problem and therefore represent a model of how location might be derived in the neocortex. We explore these ideas further in the discussion section. For our model we require (a) that the location of a feature on an object is independent of the orientation of the object, and (b) that nearby locations have similar representations. The first property allows the system to make accurate predictions when the object is sensed in novel positions relative to the body. The second property enables noise tolerance—you don't have to always sense the object in precisely the same locations.

Below we describe our neuron model, the connectivity of layers and columns, and how the sensory and location inputs are combined over time to recognize objects. A more detailed description of the activation and learning rules is available in the Materials and Methods section.

Neuron Model

We use HTM model neurons in the network (Hawkins and Ahmad, 2016). HTM neurons incorporate dendritic properties of pyramidal cells (Spruston, 2008), where proximal, basal, and apical dendritic segments have different functions (**Figure 1B**). Patterns detected on proximal dendrites represent feedforward driving input, and can cause the cell to become active. Patterns recognized on a neuron's basal and apical dendrites represent modulatory input, and will cause a dendritic spike and depolarize the cell without immediate activation. Depolarized cells fire sooner than, and thereby inhibit, non-depolarized cells that recognize the same feedforward patterns. In the rest of the paper we refer to proximal dendritic inputs as feedforward inputs, and the distal basal and apical dendritic inputs as modulatory inputs. A detailed description of functions of different dendritic integration zones can be found in Hawkins and Ahmad (2016).

Input Layer

The input layer of each cortical column consists of HTM neurons arranged in mini-columns. Here a mini-column denotes a thin vertical arrangement of neurons (Buxhoeveden, 2002). In our simulations we typically have 150–250 mini-columns per cortical column, with 16 cells per mini-column (corresponding to 2,400 to 4,000 cells). The feedforward input of cells in this layer is the sensory input. As in Hawkins and Ahmad (2016) cells within a mini-column recognize the same feedforward patterns (Jones, 2000). We map each sensory feature to a sparse set of mini-columns.

The basal modulatory input for cells in the input layer represents the location on an object. During learning, one cell in each active mini-column is chosen to learn the current location signal. During inference, cells that recognize both the modulatory location input and the feedforward driving input will inhibit other cells in the mini-column. In this way, the input layer forms a sparse representation that is unique for a specific sensory feature at a specific location on the object.

Output Layer

The output layer also contains HTM neurons. The set of active cells in the output layer represents objects. Cells in the output layer receive feedforward driver input from the input layer. During learning, the set of cells representing an object remains active over multiple movements and learns to recognize successive patterns in the input layer. Thus, an object comprises a representation in the output layer, plus an associated set of feature/location representations in the input layer.

The modulatory input to cells in the output layer comes from other output cells representing the same object, both from within the column as well as from neighboring columns via long-range lateral connections. As in the input layer, the modulatory input acts as a bias. Cells with more modulatory input will win and

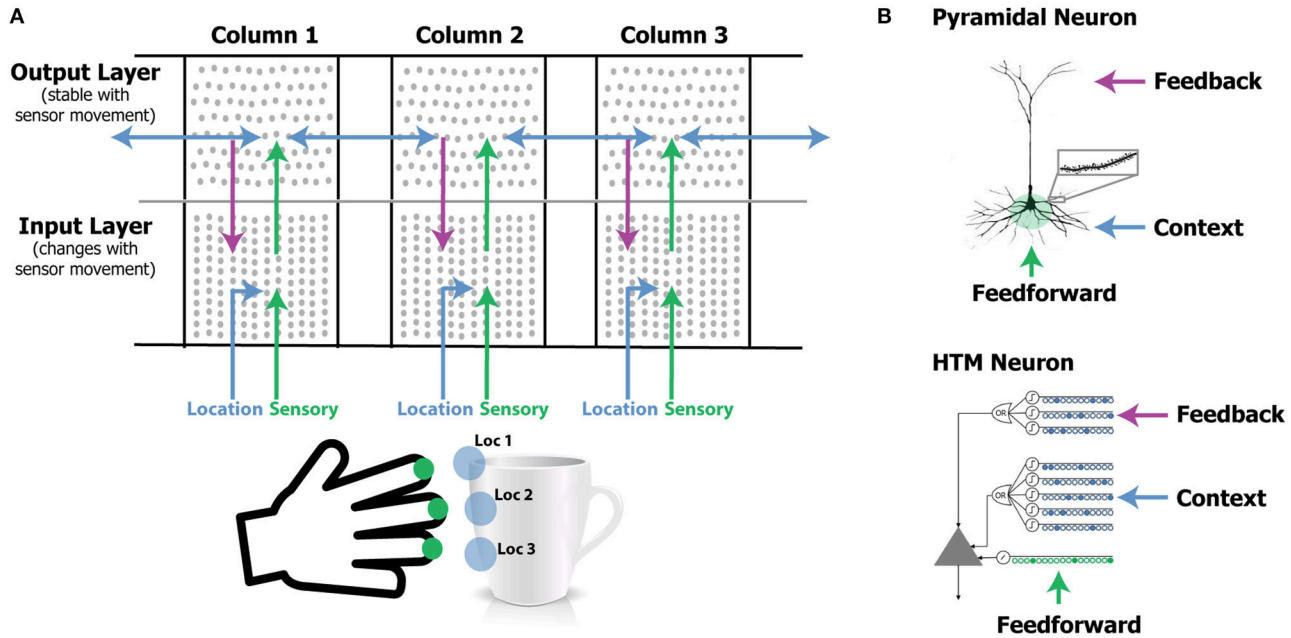


FIGURE 1 | (A) Our network model contains one or more laterally connected cortical columns (three shown). Each column receives feedforward sensory input from a different sensor array [e.g., different fingers or adjacent areas of the retina (not shown)]. The input layer combines sensory input with a modulatory location input to form sparse representations that correspond to features at specific locations on the object. The output layer receives feedforward inputs from the input layer and converges to a stable pattern representing the object (e.g., a coffee cup). Convergence in the second layer is achieved via two means. One is by integration over time as the sensor moves relative to the object, and the other is via modulatory lateral connections between columns that are simultaneously sensing different locations on the same object (blue arrows in upper layer). Feedback from the output layer to the input layer allows the input layer to predict what feature will be present after the next movement of the sensor. **(B)** Pyramidal neurons have three synaptic integration zones, proximal (green), basal (blue), and apical (purple). Although individual synaptic inputs onto basal and apical dendrites have a small impact on the soma, co-activation of a small number of synapses on a dendritic segment can trigger a dendritic spike (*top right*). The HTM neuron model incorporates active dendrites and multiple synaptic integration zones (*bottom*). Patterns recognized on proximal dendrites generate action potentials. Patterns recognized on the basal and apical dendrites depolarize the soma without generating an action potential. Depolarization is a predictive state of the neuron. Our network model relies on these properties and our simulations use HTM neurons. A detailed walkthrough of the algorithm can be found in the Supplementary Video.

inhibit cells with less modulatory input. Cells representing the same object will positively bias each other. Thus, if a column has feedforward support for objects A and B at time t, and feedforward support for objects B and C at time t+1, the output layer will converge onto the representation for object B at time t+1 due to modulatory input from time t. Similarly, if column 1 has feedforward support for objects A and B, and column 2 has feedforward support for objects B and C, the output layer in both columns will converge onto the representation for object B.

Feedback Connections

Neurons in the input layer receive feedback connections from the output layer. Feedback input representing an object, combined with modulatory input representing an anticipated new location due to movement, allows the input layer to more precisely predict the next sensory input. In our model, feedback is an optional component. If included, it improves robustness to sensory noise and ambiguity of location.

Illustrative Example

Figure 2 illustrates how the two layers of a single cortical column cooperate to disambiguate objects that have shared features, in this case a cube and a wedge. The first sensed feature-location, labeled $\mathbf{f_1}$, is ambiguous, as it could be part of either object. Therefore, the output layer simultaneously invokes a union of representations, one for each object that has that feature at that location. Feedback from the output layer to the input layer puts cells in a predictive state (shown in red). The predicted cells represent the set of all feature-locations consistent with the set of objects active in the output layer. The red cells thus represent the predictions of the network consistent with the sensations up to this point. Upon the second sensation, labeled $\mathbf{f_2}$, only the subset of cells that is consistent with these predictions and the new feature become active. Each subsequent sensation narrows down the set until only a single object is represented in the output layer. A detailed walkthrough of the algorithm can be found in the Supplementary Video.

Learning

Learning is based on simple Hebbian-style adaptation: when cells fire, previously active synapses are strengthened and inactive ones are weakened. There are two key differences with most other neural models. First, learning is isolated

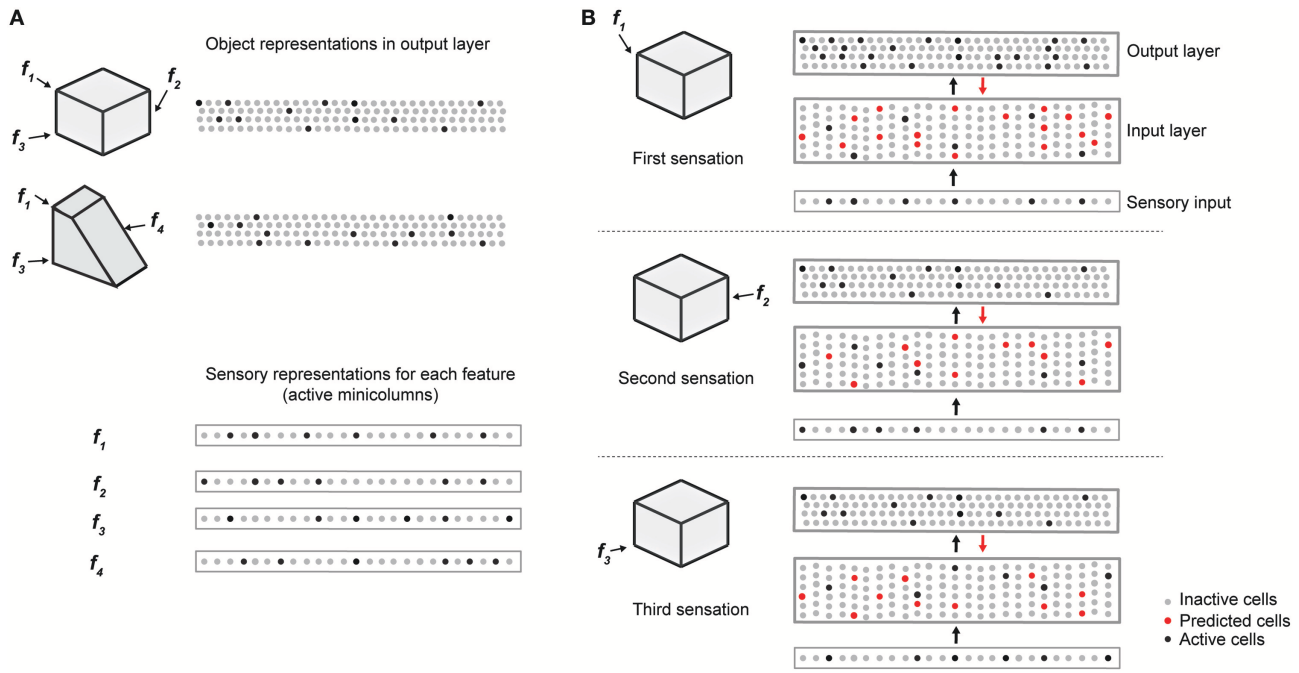


FIGURE 2 | Cellular activations in the input and output layers of a single column during a sequence of touches on an object. **(A)** Two objects (cube and wedge). For each object, three feature-location pairs are shown ($\mathbf{f}_1$ and $\mathbf{f}_3$ are common to both the cube and wedge). The output layer representations associated with each object, and the sensory representations for each feature are shown. **(B)** Cellular activations in both layers caused by a sequence of three touches on the cube (in time order from top to bottom). The first touch (at $\mathbf{f}_1$) results in a set of active cells in the input layer (black dots in input layer) corresponding to that feature-location pair. Cells in the output layer receive this representation through their feed-forward connections (black arrow). Since the input is ambiguous, the output layer forms a representation that is the union of both the cube and the wedge (black dots in output layer). Feedback from the output layer to the input layer (red arrow) causes all feature-location pairs associated with both potential objects to become predicted (red dots). The second touch (at $\mathbf{f}_2$) results in a new set of active cells in the input layer. Since $\mathbf{f}_2$ is not shared with the wedge, the representation in the output layer is reduced to only the cube. The set of predicted cells in the input layer is also reduced to the feature-location pairs of the cube. The third touch (at $\mathbf{f}_3$) would be ambiguous on its own, however, due to the past sequence of touches and self-reinforcement in the output layer, the representation of the object in the output layer remains unique to the cube. Note the number of cells shown is unrealistically small for illustration clarity.

to individual dendritic segments (Stuart and Häusser, 2001; Losonczy et al., 2008). Second, the model neuron learns by growing and removing synapses from a pool of potential synapses (Chklovskii et al., 2004). We model the growth and removal of synapses by incrementing or decrementing a variable we call "permanence." The efficacy, or weight, of a synapse is binary based on a threshold of permanence. Thus, how fast the system learns and how long memory is retained can be adjusted independent of the weight of synapses. A complete description of the biological motivation can be found in Hawkins and Ahmad (2016). Below we briefly describe how these principles enable the network to learn; the formal learning rules are described in the Materials and Methods section.

The input layer learns specific feature/location combinations. If the current feature/location combination has not been previously learned (no cell is predicted), then one cell from each active mini-column is chosen as the winner and becomes active. The winning cell is chosen as the cell with the best modulatory input match via random initial conditions. Each winner cell learns by forming and strengthening modulatory connections with the current location input. If the location input is encountered again the corresponding set of cells will be predicted. If the expected sensory feature arrives, the predicted cells will fire first, and the corresponding modulatory inputs will be reinforced. Apical dendrites of the winning cells form connections to active cells in the output layer.

The output layer learns representations corresponding to objects. When the network first encounters a new object, a sparse set of cells in the output layer is selected to represent the new object. These cells remain active while the system senses the object at different locations. Feed forward connections between the changing active cells in the input layer and unchanging active cells in the output layer are continuously reinforced. Thus, each output cell pools over multiple feature/location representations in the input layer. Dendritic segments on cells in the output layer learn by forming lateral modulatory connections to active cells within their own column, and to active cells in nearby columns.

During training, we reset the output layer when switching to a new object. In the brain, there are several ways the equivalent of a reset could occur, including a sufficiently long period of time with no sensation. When a new object is learned we select the object representation based on best match via random initial connectivity.

SIMULATION RESULTS

In this section, we describe simulation results that illustrate the performance of our network model. The network structure consists of one or more cortical columns, each with two layers, as described earlier (**Figure 1**). In the first set of simulations the input layer of each column consists of 150 mini-columns, with 16 cells per mini-column, for a total of 2,400 cells. The output layer of each column consists of 4,096 cells, which are not arranged in mini-columns. The output layer contains inter-column and intra-column connections via the distal basal dendrites of each cell. The output layer also projects back to the apical dendrites of the input layer within the same column. All connections are continuously learned and adjusted during the training process.

We trained the network on a library of up to 500 objects (**Figure 2A**). Each object consists of 10 sensory features chosen from a library of 5 to 30 possible features. Each feature is assigned a corresponding location on the object. Note that although each object consists of a unique set of features/locations, any given feature or feature/location is shared across several objects. As such, a single sensation by a single column is insufficient to unambiguously identify an object.

The set of active cells in the output layer represents the objects that are recognized by the network. During inference we say that the network unambiguously recognizes an object when the representation of the output layer overlaps significantly with the representation for correct object and not for any other object. Complete details of object construction and recognition are described in Materials and Methods.

In the following paragraphs we first describe network convergence, using single and multi-column networks. We then discuss the capacity of the network.

Network Convergence

As discussed earlier, the representation in the output layer is consistent with the recent sequence of sensed features and locations. Multiple output representations will be active simultaneously if the sensed features and locations are not unique to one particular object. The output converges to a single object representation over time as the object is explored via movement. **Figure 3** illustrates the rate of convergence for a one column network and for a three-column network. Multiple columns working together reduces the number of sensations needed for recognition.

In **Figure 4A** we plot the mean number of sensations required to unambiguously recognize an object as a function of the total number of objects in the training set. As expected, the number of sensations required increases with the total number of stored objects. However, in all cases the network eventually correctly recognizes every object. The number of sensations is also dependent on the overall confusion between the set of objects. The more unique the objects, the faster the network can disambiguate them.

Figure 4B illustrates the mean number of sensations required to recognize an object as a function of the number of cortical columns in the network. The graph demonstrates the advantage of including multiple columns. The number of sensations required decreases rapidly as the number of columns increases. Thus, although single column networks can recognize the objects, multicolumn networks are much faster. With a sufficient number of columns, the network disambiguates even highly confusing objects with a single sensation. In this experiment, each column receives lateral input from every other column.

In **Figure 4C** we plot the fraction of objects that can be unambiguously recognized ("accuracy") as a function of the number of sensations. We compare a single column network to an ideal observer model with and without location (see Materials and Methods). The performance of our model is close to the ideal observer with locations. It takes many more sensations for the model without locations to recognize the objects, and some objects cannot be distinguished (this is not shown on the graph as we plot the average number of sensations), underscoring the importance of the location signal. We have also shown that multi-column networks perform close to an ideal observer model that similarly observes multiple features per sensation (Supplementary Figure 9). Taken together, these results show that our biologically derived sensorimotor network operates close to the non-biological ideal model with respect to accuracy and speed of convergence.

Capacity

In the network model presented here, each cortical column builds predictive models of objects. A key question is, how many objects can a single column represent? Also, does adding more columns impact capacity? In this section we explore the effect of various parameters on the number of objects that can be accurately recognized. We define capacity as the maximum number of objects a network can learn and recognize without confusion. We analyze four different factors that impact capacity: the representational space of the network, the number of mini-columns in the input layer, the number of neurons in the output layer, and the number of cortical columns. In our analysis we used numbers similar to those reported in experimental data. For example, cortical columns vary from 300 μm to 600 μm in diameter (Mountcastle, 1997), where the diameter of a mini-column is estimated to be in the range of 30–60 μm (Buxhoeveden, 2002). For our analysis and simulations we assumed a cortical column contains between 150 and 250 mini-columns.

First, the neural representation must allow the input and output layers to represent large numbers of unique feature/locations and objects. As illustrated in **Figure 2**, both layers use sparse representations. Sparse representations have several attractive mathematical properties that allow robust representation of a very large number of elements (Ahmad and Hawkins, 2016). With a network of 150 mini-columns, 16 cells per mini-column, and 10 simultaneously active mini-columns, we can uniquely represent $\binom{150}{10} \sim 10^{15}$ sensory features. Each feature can be represented at 16^{10} unique locations. Similarly, the output layer can represent $\binom{n}{w}$ unique objects, where n is the number of output cells and w is the number of active cells at any time. With such large representational

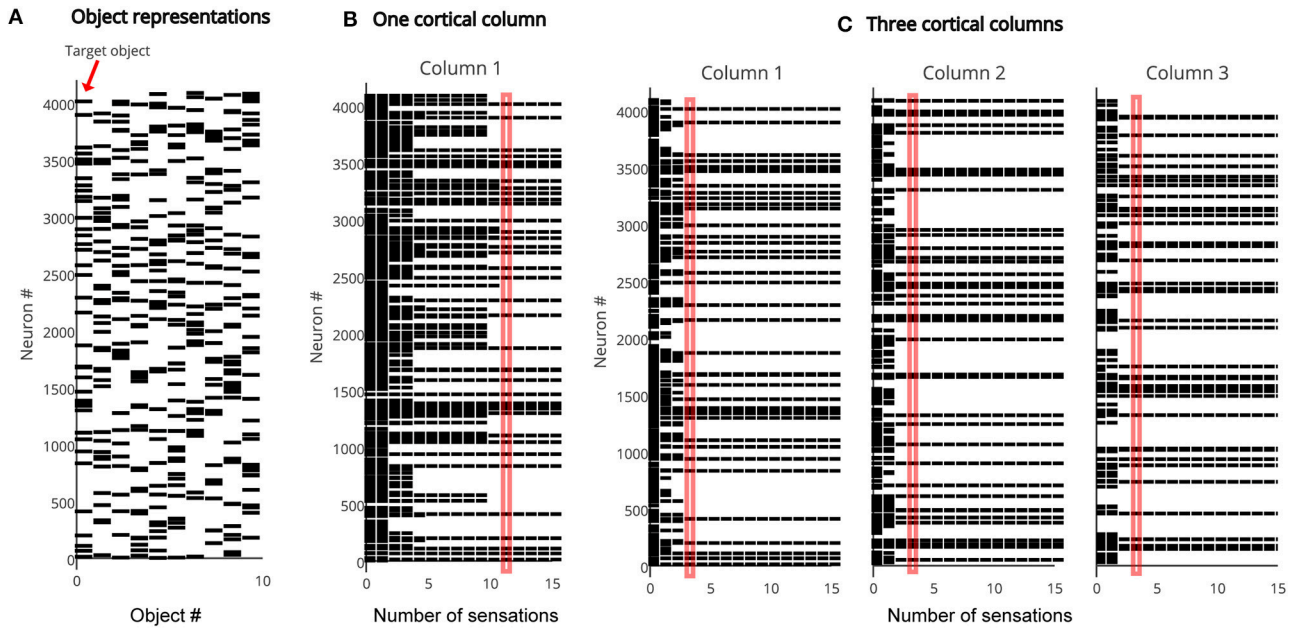


FIGURE 3 | (A) The output layer represents each object by a sparse pattern. We tested the network on the first object. **(B)** Activity in the output layer of a single column network as it touches the object. The network converges after 11 sensations (red rectangle). **(C)** Activity in the output layer of a three column network as it touches the object. The network converges much faster, after four sensations (red rectangle). In both **(B,C)** the representation in Column 1 is the same as the target object representation after convergence.

spaces, it is extremely unlikely for two feature/location pairs or two object representations to have a significant number of overlapping bits by chance (Supplementary Material). Therefore, the number of objects and feature location pairs that can be uniquely represented is not a limiting factor in the capacity of the network.

As the number of learned objects increases, neurons in the output layer form increasing numbers of connections to neurons in the input layer. If an output neuron connects to too many input neurons, it may be falsely activated by a pattern it was not trained on. Therefore, the capacity of the network is limited by the pooling capacity of the output layer. Mathematical analysis suggests that a single cortical column can store hundreds of objects before reaching this limit (see Supplementary Material).

To measure actual network capacity, we trained networks with an increasing number of objects and plotted recognition accuracy. For a single cortical column, with 4,096 cells in the output layer and 150 mini-columns in the input layer, the recognition accuracy remains perfect up to 400 objects (**Figure 5A**, blue). The retrieval accuracy drops when the number of learned objects exceeds the capacity of the network.

From the mathematical analysis, we expect the capacity of the network to increase as the size of the input and output layers increase. We again tested our analysis through simulations. With the number of active cells fixed, the capacity increases with the number of mini-columns in the input layer (**Figure 5A**). This is because with more cells in the input layer, the sparsity of activation increases, and it is less likely for an output cell to be falsely activated. The capacity also significantly increases with the number of output cells when the size of the input layer is fixed (**Figure 5B**). This is because the number of feedforward connections per output cell decreases when there are more output cells available. We found that if the size of individual columns is fixed, adding columns can increase capacity (**Figure 5C**). This is because the lateral connections in the output layer can help disambiguate inputs once individual cortical columns hit their capacity limit. However, this effect is limited; the incremental benefit of additional columns decreases rapidly.

The above simulations demonstrate that it is possible for a single cortical column to model and recognize several hundred objects. Capacity is most impacted by the number of cells in the input and output layers. Increasing the number of columns has a marginal effect on capacity. The primary benefit of multiple columns is to dramatically reduce the number of sensations needed to recognize objects. A network with one column is like looking at the world through a straw; it can be done, but slowly and with difficulty.

Noise Robustness

We evaluated robustness of a single column network to noise. After the network learned a set of objects, we added varying amounts of random noise to the sensory and location inputs. The noise affected the active bits in the input without changing its overall sparsity (see Materials and Methods). Recognition accuracy after 30 touches is plotted as a function of noise (**Figure 6A**). There is no impact on the recognition accuracy up

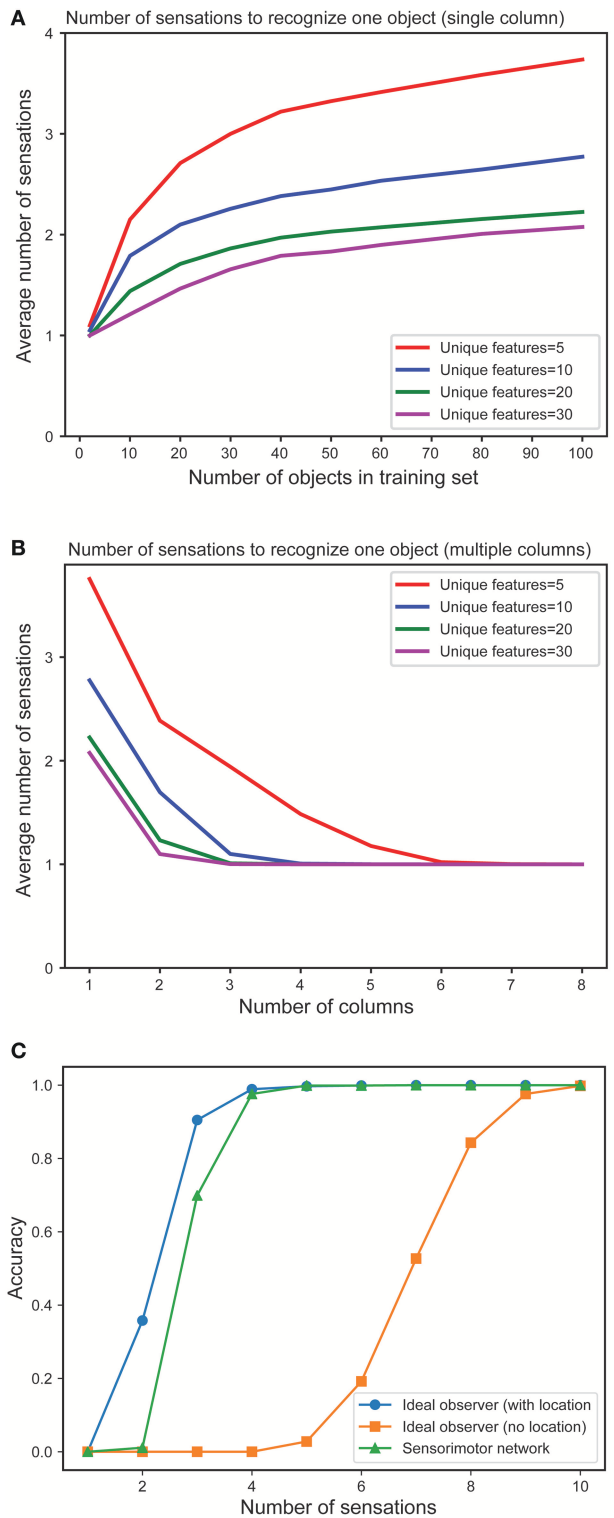


FIGURE 4 | (A) Mean number of sensations needed to unambiguously recognize an object with a single column network as the set of learned objects increases. We train models on varying numbers of objects, from 1 to 100 and plot the average number of sensations required to unambiguously recognize a single object. The different curves show how convergence varies with the total *(Continued)*

FIGURE 4 | Continued
number of unique features from which objects are constructed. In all cases the network eventually recognizes the object. Recognition requires fewer sensations when the set of features is greater. **(B)** Mean number of observations needed to unambiguously recognize an object with multi-column networks as the set of columns increases. We train each network with 100 objects and plot the average number of sensations required to unambiguously recognize an object. The required number of sensations rapidly decreases as the number of columns increases, eventually reaching one. **(C)** Fraction of objects that can be unambiguously recognized as a function of number of sensations for an ideal observer model with location (blue), without location (orange) and our one-column sensorimotor network (green).

to 20% noise in the sensory input and 40% noise in the location input. We also found that the convergence speed was impacted by noise in the location input (**Figure 6B**). It took more sensations to recognize the object when the location input is noisy.

MAPPING TO BIOLOGY

Anatomical evidence suggests that the sensorimotor inference model described above exists at least once in each column (layers 4 and 2/3) and perhaps twice (layers 6a and 5). We adopt commonly used terminology to describe these layers. This is a convenience as the connectivity and physiology of cell populations is what matters. Cells we describe as residing in separate layers may actually intermingle in cortical tissue (Guy and Staiger, 2017).

Layers 4 and 2/3

The primary instance of the model involves layers 4 and 2/3 as illustrated in **Figure 7A**. The following properties evident in L4 and L2/3 match our model. L4 cells receive direct thalamic input from sensory "core" regions (e.g., LGN; Douglas and Martin, 2004). This input onto proximal dendrites exhibits driver properties (Viaene et al., 2011a). L4 cells do not form long range connections within their layer (Luhmann et al., 1990). L4 cells project to and activate cells in L2/3 (Lohmann and Rörig, 1994; Feldmeyer et al., 2002; Sarid et al., 2007), and receive feedback from L2/3 (Lefort et al., 2009; Markram et al., 2015). L2/3 cells project long distances within their layer (Stettler et al., 2002; Hunt et al., 2011) and are also a major output of cortical columns (Douglas and Martin, 2004; Shipp, 2007). It is known that L2/3 activation follows L4 activation (Constantinople and Bruno, 2013).

The model predicts that a representation of location is input to the basal distal dendrites of the input layer. A timing requirement of our model is that the location signal is a predictive signal that must precede the arrival of the sensory input. This is illustrated by the red line in **Figure 7A**. About 45% of L4 synapses come from cells in L6a (Binzegger et al., 2004). The axon terminals were found to show a strong preference for contacting basal dendrites (McGuire et al., 1984) and activation of L6a cells caused weak excitation of L4 cells (Kim et al., 2014). Therefore, we propose that the location representation needed for the upper model comes from L6a.

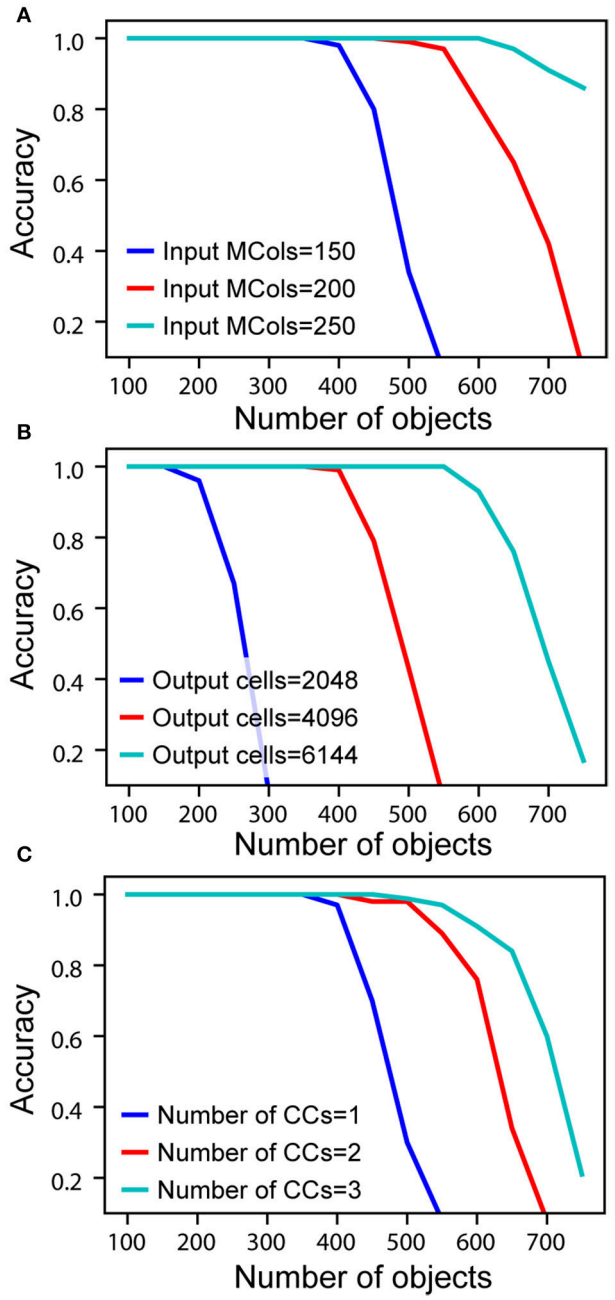


FIGURE 5 | Recognition accuracy is plotted as a function of the number of learned objects. **(A)** Network capacity relative to number of mini-columns in the input layer. The number of output cells is kept at 4,096 with 40 cells active at any time. **(B)** Network capacity relative to number of cells in the output layer. The number of active output cells is kept at 40. The number of mini-columns in the input layer is 150. **(C)** Network capacity for one, two, and three cortical columns (CCs). The number of mini-columns in the input layer is 150, and the number of output cells is 4,096.

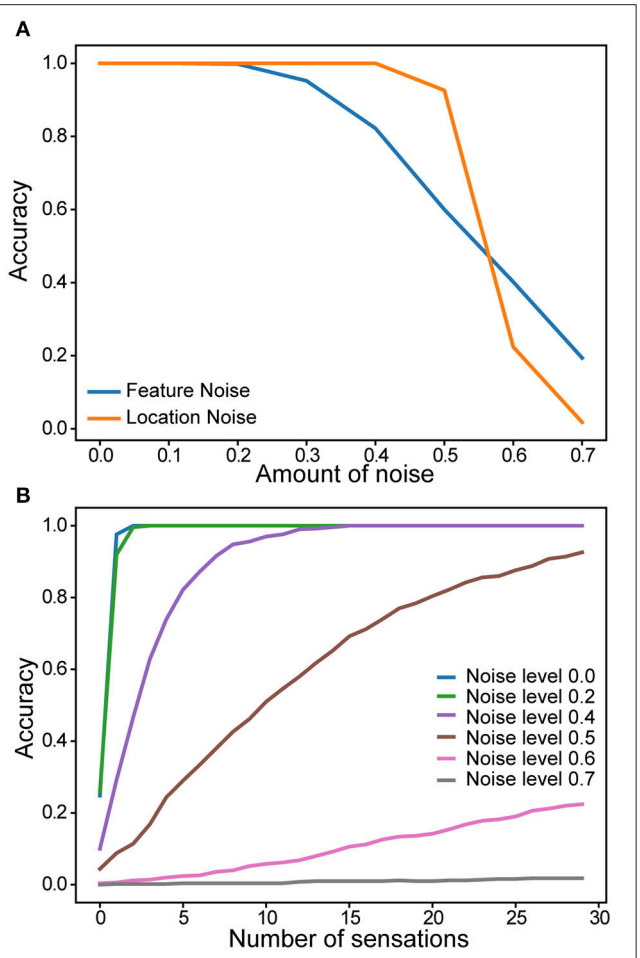


FIGURE 6 | Robustness of a single column network to noise. **(A)** Recognition accuracy is plotted as a function of the amount of noise in the sensory input (blue) and in the location input (yellow). **(B)** Recognition accuracy as a function of the number of sensations. Colored lines correspond to noise levels in the location input.

Layers 6a and 5

Another potential instance of the model is in layers 6a and 5 as illustrated in **Figure 7B**. The following properties evident in L6a and L5 match our model. L6a cells receive direct thalamic input from sensory "core" regions (e.g., LGN; Thomson, 2010). This input exhibits driver properties and resembles the thalamocortical projections to L4 (Viaene et al., 2011b). L6a cells project to and activate cells in L5 (Thomson, 2010). Recent experimental studies found that the axons of L6 CT neurons densely ramified within layer 5a in both visual and somatosensory cortices of the mouse, and activation of these neurons generated large excitatory postsynaptic potentials (EPSPs) in pyramidal neurons in layer 5a (Kim et al., 2014). L6a cells receive feedback from L5 (Thomson, 2010). L5 cells project long distances within their layer (Schnepel et al., 2015) and L5 cells are also a major output of cortical columns (Douglas and Martin, 2004; Guillery and Sherman, 2011; Sherman and Guillery, 2011). There are three types of pyramidal neurons in L5 (Kim et al., 2015). Here we are referring to only one of them, the larger neurons with thick apical trunks that send an axon branch to relay cells in the thalamus (Ramaswamy and Markram, 2015). However, there is also empirical evidence our model does

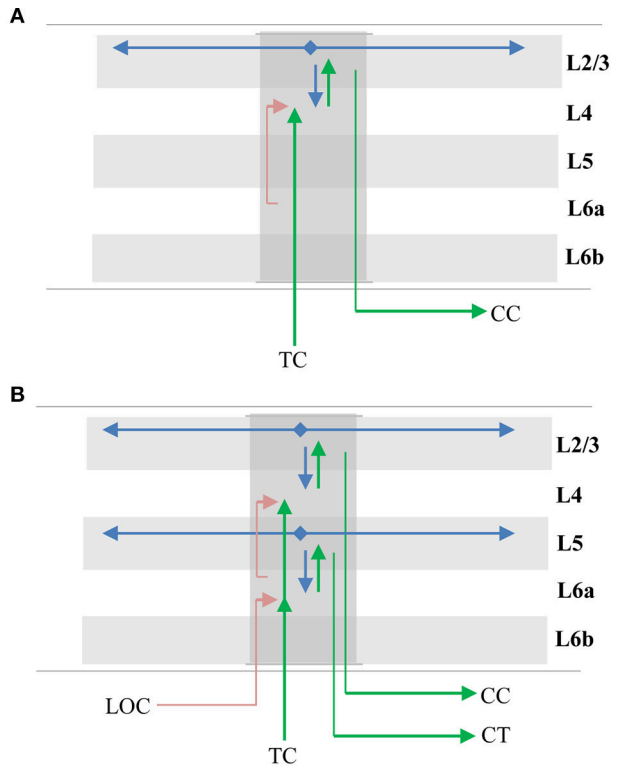


FIGURE 7 | Mapping of sensorimotor inference network onto experimentally observed cortical connections. Arrows represent documented pathways. **(A)** First instance of network; L4 is input layer, L2/3 is output layer. Green arrows are feedforward pathway, from thalamo-cortical (TC) relay cells, to L4, to L2/3 cortico-cortical (CC) output cells. Cells in L2/3 also project back to L4 and to adjacent columns (blue arrows); these projections depolarize specific sets of cells that act as predictions (see text). Red arrow is location signal originating in L6a and terminating on basal distal dendrites of L4 cells. **(B)** Possible second instance of network; L6a is input layer, L5 is output layer. Both instances of the network receive feedforward input from the same TC axons, thus the two networks run in parallel (Constantinople and Bruno, 2013; Markov et al., 2013). The origin and derivation of the location signal (LOC) is unknown but likely involves local processing as well as input from other regions (see text and Discussion). The output of the upper network makes direct cortical-cortical (CC) connections, whereas the output of the lower network projects to thalamic relay cells before projecting to the next region.

not map cleanly to L6a and L5. For example, Constantinople and Bruno (2013) have shown a sensory stimulus will often cause L5 cells to fire simultaneously or even slightly before L6 cells, which is inconsistent with the model. Therefore, whether L6a and L5 can be interpreted as an instance of the model is unclear.

Origin of Location Signal

The derivation of the location representation in L6a is unknown. Part of the answer will involve local processing within the lower layers of the column and part will likely involve long range connections between corresponding regions in "what" and "where" pathways (Thomson, 2010). Parallel "what" and "where" pathways exist in all the major sensory modalities (Ungerleider and Haxby, 1994; Ahveninen et al., 2006). Evidence suggests that regions in "what" pathways form representations that exhibit increasing invariance to translation, rotation or scale and increasing selectivity to sensory features in object centered coordinates (Rust and DiCarlo, 2010). This effect can be interpreted as forming allocentric representations. In contrast, it has been proposed that regions in "where" pathways form representations in egocentric coordinates (Goodale and Milner, 1992). If an egocentric motor behavior is generated in a "where" region, then a copy of the motor command will need to be sent to the corresponding "what" region where it can be converted to a new predicted allocentric location. The conversion is dependent on the current position and orientation of the object relative to the body. It is for this reason we suggest that the origin of the location signal might involve long-range connections between "where" and "what" regions. In the Discussion section we will describe how the location might be generated.

Physiological Evidence

In addition to anatomical support, there are several physiological predictions of the model that are supported by empirical observation. L4 and L6a cells exhibit "simple" receptive fields (RFs) while L2/3 and L5 cells exhibit "complex" RFs (Hubel and Wiesel, 1962; Gilbert, 1977). Key properties of complex cells include RFs influenced by a wider area of sensory input and increased temporal stability (Movshon et al., 1978). L2/3 cells have receptive fields that are twice the size of L4 cells in the primary somatosensory cortex (Chapin, 1986). A distinct group of cells with large and non-oriented receptive fields were found mostly in layer 5 of the visual cortex (Mangini and Pearlman, 1980; Lemmon and Pearlman, 1981). These properties are consistent with, and observed, in the output layer of our model.

The model predicts that cells in a mini-column in the input layer (L4 and L6a) will have nearly identical RFs when presented with an input than cannot be predicted as part of a previously learned object. However, in the context of learned objects, the cells in a mini-column will differentiate. One key differentiation is that individual cells will respond only in specific contexts. This differentiation has been observed in multiple modalities (Vinje and Gallant, 2002; Yen et al., 2006; Martin and Schröder, 2013; Gavornik and Bear, 2014). Our model is also consistent with findings that early sensory areas are biased toward recent perceptual recognition results (St. John-Saaltink et al., 2016).

A particularly relevant version of this phenomenon is "border ownership" (Zhou et al., 2000). Cells which have similar classic receptive fields when presented with isolated edge-like features, diverge, and fire uniquely when the feature is part of a larger object. Specifically, the cells fire when the feature is at a particular location on a complex object, a behavior predicted and exhibited by our model. To explain border ownership, researchers have proposed a layer of cells that perform "grouping" of inputs. The grouping cells are stable over time (Craft et al., 2007). The output layer of our model performs this function. "Border ownership" is a form of complex object modeling. It has been observed in both primary and secondary sensory regions (Zhou et al., 2000). We predict that similar properties can be observed in

primary and secondary sensory regions for even more complex and three-dimensional objects.

Lee et al. show that enhancement of motor cortex activity facilitates sensory-evoked responses of topographically aligned neurons in primary somatosensory cortex (Lee et al., 2008). Specifically, they found that S1 corticothalamic neurons in whisker/barrel cortex responded more robustly to whisker deflections when motor cortex activity was focally enhanced. This supports the model hypothesis that behaviorally-generated location information projects in a column-by-column fashion to primary sensory regions.

DISCUSSION

Relationship with Previous Models

Due to the development of new experimental techniques, knowledge of the laminar circuitry of the cortex continues to grow (Thomson and Bannister, 2003; Thomson and Lamy, 2007). It is now possible to reconstruct and simulate the circuitry in an entire cortical column (Markram et al., 2015). Over the years, numerous efforts have been undertaken to develop models of cortical columns. Many cortical column models aim to explain neurophysiological properties of the cortex. For example, based on their studies on the cat visual cortex (Douglas and Martin, 1991), provided one of the first canonical microcircuit models of a cortical column. This model explains intracellular responses to pulsed visual stimulations and has remained highly influential (Douglas and Martin, 2004). Hill and Tononi (2004) constructed a large-scale model of point neurons that are organized in a repeating columnar structure to explain the difference of brain states during sleep and wakefulness. Traub et al. (2004) developed a single-column network model based on multi-compartmental biophysical models to explain oscillatory, epileptic, and sleeplike phenomena. Haeusler and Maass (2007) compared cortical microcircuit models with and without the lamina-specific structure and demonstrated several computational advantages of more realistic cortical microcircuit models. Reimann et al. (2013) showed that the neocortical local field potentials can be explained by a cortical column model composed of >12,000 reconstructed multi-compartmental neurons.

Although these models provided important insights on the origin of neurophysiological signals, there are relatively few models proposing the functional roles of layers and columns. Bastos et al. (2012) discussed the correspondence between the micro-circuitry of the cortical column and the connectivity implied by predictive coding. This study used a coarse microcircuit model based on the work of Douglas and Martin (2004) and lacked recent experimental evidence and detailed connectivity patterns across columns.

Raizada and Grossberg (2003) described the LAMINART model to explain how attention might be implemented in the visual cortex. This study highlighted the anatomical connections of the L4-L2/3 network and proposed that perceptual grouping relies on long-range lateral connections in L2/3. This is consistent with our proposal of the stable object representation in L2/3. A recent theory of optimal context integration proposes that long-range lateral connections are used to optimally integrate information from the surround (Iyer and Mihalas, 2017). The structure of their model is broadly consistent with the theories presented here, and provides a possible mathematical basis for further analysis.

The Benefit of Cortical Columns

Our research has been guided by Mountcastle's definition of a cortical column (Mountcastle, 1978, 1997), as a structure "formed by many mini-columns bound together by short-range horizontal connections." The concept plays an essential role in the theory presented in this paper. Part of our theory is that each repetitive unit, or "column," of sensory cortex can learn complete objects by locally integrating sensory and location data over time. In addition, we have proposed that multiple cortical columns greatly speed up inference and recognition time by integrating information in parallel across dispersed sensory areas.

An open issue is the exact anatomical organization of columns. We have chosen to describe a model of columns with discrete inter-column boundaries. This type of well-defined structure is most clear in the rat barrel cortex (Lubke et al., 2000; Bureau et al., 2004; Feldmeyer et al., 2013) but Mountcastle and others have pointed out that although there are occasional discontinuities in physiological and anatomical properties, there is a diverse range of structures and the more general rule is continuity (Mountcastle, 1978; Horton and Adams, 2005; Rockland, 2010).

Mountcastle's concept of a repetitive functional unit, whether continuous or discrete, is useful to understand the principles of cortical function. Our model assigns a computational benefit to columns, that of integrating discontinuous information in parallel across disparate areas. This basic capability is independent of any specific type of column (such as, hypercolumns or ocular dominance columns), and independent of discrete or continuous structures. The key requirement is that each column models a different subset of sensory space and is exposed to different parts of the world as sensors move.

Generating the Location Signal

A key prediction of our model is the presence of a location signal in each column of a cortical region. We deduced the need for this signal based on the observation that cortical regions predict new sensory inputs due to movement (Duhamel et al., 1992; Nakamura and Colby, 2002; Li and DiCarlo, 2008). To predict the next sensory input, a patch of neocortex needs to know where a sensor will be on a sensed object after a movement is completed. The prediction of location must be done separately for each part of a sensor array. For example, for the brain to predict what each finger will feel on a given object, it has to predict a separate allocentric location for each finger. There are dozens of semi-independent areas of sensation on each hand, each of which can sense a different location and feature on an object. Thus, the allocentric location signals must be computed in a part of the brain where somatic topology is similarly granular. For touch, this suggests the derivation of allocentric location is occurring in each column throughout primary regions such as, S1 and S2. The same argument holds for primary visual regions, as each patch of the retina observes different parts of objects.

Although we don't know how the location signal is generated, we can list some theoretically-derived requirements. A column needs to know its current location on an object, but it also needs to predict what its new location will be after a movement is completed. To translate an egocentric motor signal into a predicted allocentric location, a column must also know the orientation of the object relative to the body part doing the moving. This can be expressed in the pseudo-equation [current location + orientation of object + movement ≥ predicted new location]. This is a complicated task for neurons to perform. Fortunately, it is highly analogous to what grid cells do. Grid cells are a proof that neurons can perform these types of transformations, and they suggest specific mechanisms that might be deployed in cortical columns.

(1) Grid cells in the entorhinal cortex (Hafting et al., 2005; Moser et al., 2008) encode the location of an *animal's body* relative to an external *environment*. A sensory cortical column needs to encode the location of a *part of the animal's body* (a sensory patch) relative to an external *object*.
(2) Grid cells use path integration to predict a new location due to movement (Kropff et al., 2015). A column must also use path integration to predict a new location due to movement.
(3) To predict a new location, grid cells combine current location, with movement, with head direction cells (Moser et al., 2014). Head direction cells represent the "orientation" of the "animal" relative to an external environment. Columns need a representation of the "orientation" of a "sensory patch" relative to an external object.
(4) The representation of space using grid cells is dimensionless. The dimensionality of the space they represent is defined by the tiling of grid cells, combined with how the tiling maps to behavior. Similarly, our model uses representations of location that are dimensionless.

These analogs, plus the fact that grid cells are phylogenetically older than the neocortex, lead us to hypothesize that the cellular mechanisms used by grid cells were preserved and replicated in the sub-granular layers of each cortical column. It is not clear if a column needs neurons that are analogous to place cells (Moser et al., 2015). Place cells are believed to associate a location (derived from grid cells) with features and events. They are believed to be important for episodic memory. Presently, we don't see an analogous requirement in cortical columns.

Today we have no direct empirical evidence to support the hypothesis of grid-cell like functionality in each cortical column. We have only indirect evidence. For example, to compute location, cortical columns must receive dynamically updated inputs regarding body pose. There is now significant evidence that cells in numerous cortical areas, including sensory regions, are modulated by body movement and position. Primary visual and auditory regions contain neurons that are modulated by eye position (Trotter and Celebrini, 1999; Werner-Reiss et al., 2003) as do areas MT, MST, and V4 (Bremmer, 2000; DeSouza et al., 2002). Cells in frontal eye fields (FEF) respond to auditory stimuli in an eye-centered frame of reference (Russo and Bruce, 1994). Posterior parietal cortex (PPC) represents multiple frames of reference including head-centered (Andersen et al., 1993) and body-centered (Duhamel et al., 1992; Brotchie et al., 1995, 2003; Bolognini and Maravita, 2007) representations. Motor areas also contain a diverse range of reference frames, from representations of external space independent of body pose to representations of specific groups of muscles (Graziano and Gross, 1998; Kakei et al., 2003). Many of these representations are granular, specific to particular body areas, and multisensory, implying numerous transformations are occurring in parallel (Graziano et al., 1997; Graziano and Gross, 1998; Rizzolatti et al., 2014). Some models have shown that the above information can be used to perform coordinate transformations (Zipser and Andersen, 1988; Pouget and Snyder, 2000).

Determining how columns derive the allocentric location signal is a current focus of our research.

Role of Inhibitory Neurons

There are several aspects of our model that require inhibition. In the input layer, neurons in mini-columns mutually inhibit each other. Specifically, neurons that are partially depolarized (in the predictive state) generate a first action potential slightly before cells that are not partially depolarized. Cells that spike first prevent other nearby cells from firing. This requires a very fast, winner-take-all type of inhibition among nearby cells, and suggests that such fast inhibitory neurons contain stimulus-related information, which is consistent with recent experiment findings (Reyes-Puerta et al., 2015a,b). Simulations of the timing requirement for this inhibition can be found in Billaudelle and Ahmad (2015). Activations in the output layer do not require very fast inhibition. Instead, a broad inhibition within the layer is needed to maintain the sparsity of activation patterns. Experiment evidence for both fast and broad inhibition have been reported in the literature (Helmstaedter et al., 2009; Meyer et al., 2011).

Our simulations do not model inhibitory neurons as individual cells. The functions of inhibitory neurons are encoded in the activation rules of the model. A more detailed mapping to specific inhibitory neuron types is an area for future research.

Hierarchy

The neocortex processes sensory input in a series of hierarchically arranged regions. As input ascends from region to region, cells respond to larger areas of the sensory array and to more complex features. A common assumption is that complete objects can only be recognized at a level in the hierarchy where cells respond to input over the entire sensory array.

Our model proposes an alternate view. All cortical columns, even columns in primary sensory regions, are capable of learning representations of complete objects. However, our network model is limited by the spatial extent of the horizontal connections in the output layer. Therefore, hierarchy is still required in many situations. For example, say we present an image of a printed letter on the retina. If the letter occupies a small part of the retina, then columns in V1 could recognize the letter. If, however, the letter is expanded to occupy a large part of the retina, then columns in V1 would no longer be able to recognize the letter because the features that define the letter

are too far apart to be integrated by the horizontal connections in L2/3. In this case, a converging input onto a higher cortical region would be required to recognize the letter. Thus, the cortex learns multiple models of objects, both within a region and across hierarchical levels.

What would occur if multiple objects were being sensed at the same time? In our model, one part of a sensory array could be sensing one object and another part of the sensory array could be sensing a different object. Difficulty would arise if the sensations from two or more objects were overlaid or interspersed on a region, such as, if your index and ring finger touched one object while your thumb and middle finger touched another object. In these situations, we suspect the system would settle on one interpretation or the other.

Sensory information is processed in parallel pathways, sometimes referred to as "what" and "where" pathways. We propose that our object recognition model exists in "what" regions, which are associated with the ability to recognize objects. How might we interpret "where" pathways in light of our model? First, the anatomy in the two pathways is similar. This suggests that "what" and "where" regions perform similar operations, but achieve different results by processing different types of data. For example, our network might learn models of ego-centric space if the location signal represented ego-centric locations. Second, we suspect that bi-directional connections between what and where regions are required for converting ego-centric motor behaviors into allocentric locations. We are currently exploring these ideas.

Vision, Audition, and Beyond

We described our model using somatic sensation. Does it apply to other sensory modalities? We believe it does. Consider vision. Vision and touch are both based on an array of receptors topologically mapped to an array of cortical columns. The retina is not like a camera. The blind spot and blood vessels prevent all parts of an object from being sensed simultaneously, and the density of receptors in the retina is not uniform. Similarly, the skin cannot sense all parts of an object at once, and the distribution of somatic receptors is not uniform. Our model is indifferent to discontinuities and non-uniformities. Both the skin and retina move, exposing cortical columns to different parts of sensed objects over time. The methods for determining the allocentric location signal for touch and vision would differ somewhat. Somatic sensation has access to richer proprioceptive inputs, whereas vision has access to other clues such as, ocular disparity. Aside from differences in how allocentric location is determined, our model is indifferent to the underlying sensory modality. Indeed, columns receiving visual input could be interspersed with columns receiving somatic input, and the long-range intercolumn connections in our model would unite these into a single object representation.

Similar parallels can be made for audition. Perhaps the more powerful observation is that the anatomy supporting our model exists in most, if not all, cortical regions. This suggests that no matter what kind of information a region is processing, its feedforward input is interpreted in the context of a location. This would apply to high-level concepts as well as low-level sensory data. This hints at why it is easier to memorize a list of items when they are mentally associated with physical locations, and why we often use mental imagery to convey abstract concepts.

Testable Predictions

A number of experimentally testable predictions follow from this theory.

(1) The theory predicts that sensory regions will contain cells that are stable over movements of a sensor while sensing a familiar object.
(2) The set of stable cells will be both sparse and specific to object identity. The cells that are stable for a given object will in general have very low overlap with those that are stable for a completely different object.
(3) Layers 2/3 of cortical columns will be able to independently learn and model complete objects. We expect that the complexity of the objects a column can model will be related to the extent of long-range lateral connections.
(4) Activity within the output layer of each cortical column (layers 2/3) will become sparser as more evidence is accumulated for an object. Activity in the output layer will be denser for ambiguous objects. These effects will only be seen when the animal is freely observing familiar objects.
(5) These output layers will form stable representations. In general, their activity will be more stable than layers without long-range connections.
(6) Activity within the output layers will converge on a stable representation slower with long-range lateral connections disabled, or with input to adjacent columns disabled.
(7) The theory provides an algorithmic explanation for border ownership cells (Zhou et al., 2000). In general each region will contain cells tuned to the location of features in the object's reference frame. We expect to see these representations in layer 4.

Summary

Our research has focused on how the brain makes predictions of sensory inputs. Starting with the premise that all sensory regions make predictions of their constantly changing input, we deduced that each small area in a sensory region must have access to a location signal that represents where on an object the column is sensing. Building on this idea, we deduced the probable function of several cellular layers and are beginning to understand what cortical columns in their entirety might be doing. Although there are many things we don't understand, the big picture is increasingly clear. We believe each cortical column learns a model of "its" world, of what it can sense. A single column learns the structure of many objects and the behaviors that can be applied to those objects. Through intra-laminar and long-range cortical-cortical connections, columns that are sensing the same object can resolve ambiguity.

In 1978 Vernon Mountcastle reasoned that since the complex anatomy of cortical columns is similar in all of the neocortex, then all areas of the neocortex must be performing a similar function (Mountcastle, 1978). His hypothesis remains controversial partly because we haven't been able to identify what functions a cortical column performs, and partly because it has

been hard to imagine what single complex function is applicable to all sensory and cognitive processes.

The model of a cortical column presented in this paper is described in terms of a sensory regions and sensory processing, but the circuitry underlying our model exists in all cortical regions. Thus, if Mountcastle's conjecture is correct, even high-level cognitive functions, such as, mathematics, language, and science would be implemented in this framework. It suggests that even abstract knowledge is stored in relation to some form of "location" and that much of what we consider to be "thought" is implemented by inference and behavior generating mechanisms originally evolved to move and infer with fingers and eyes.

MATERIALS AND METHODS

Here we formally describe the activation and learning rules for the HTM sensorimotor inference network. We use a modified version of the HTM neuron model (Hawkins and Ahmad, 2016) in the network. There are three basic aspects of the algorithm: initialization, computing cell states, and learning. These steps are described along with implementation and simulation details.

Notation

Let N^{in} represent the number of mini-columns in the input layer, M the number of cells per mini-column in the input layer, N^{out} the number of cells in the output layer and N^c the number of cortical columns. The number of cells in the input layer and output layer is MN^{in} and N^{out}, respectively, for each cortical column. Each input cell receives both the sensory input and a contextual input that corresponds to the location signal. The location signal is a N^{ext} dimensional sparse vector $\boldsymbol{L}$.

Each cell can be in one of three states: active, predictive, or inactive. We use $M \times N^{in}$ binary matrices $\mathbf{A}^{in}$ and $\mathbf{\Pi}^{in}$ to denote activation state and predictive state of input cells and use the N^{out} dimensional binary vector $\mathbf{A}^{out}$ to denote the activation state of the output cells in a cortical column. The concatenated output of all cortical columns is represented as a $N^{out}N^{column}$ dimensional binary vector $\bar{\mathbf{A}}^{out}$. At any point in time there are only a small number of cells active, so these are generally very sparse.

Each cell maintains a single proximal dendritic segment and a set of basal distal dendritic segments (denoted as basal below). Proximal segments contain feedforward connections to that cell. Basal segments represent contextual input. The contextual input acts as a tiebreaker and biases the cell to win. The contextual input to a cell in the input layer is a vector representing the external location signal $\boldsymbol{L}$. The contextual input to a cell in the output layer comes from other output cells in the same or different cortical columns.

For each dendritic segment, we maintain a set of "potential" synapses between the dendritic segment and other cells that could potentially form a synapse with it (Chklovskii et al., 2004; Hawkins and Ahmad, 2016). Learning is modeled by the growth of new synapses from this set of potential synapses. A "permanence" value is assigned to each potential synapse and represents the growth of the synapse. Potential synapses are represented by permanence values greater than zero. A permanence value close to zero represents an unconnected synapse that is not fully grown. A permanence value greater than the connection threshold represents a connected synapse. Learning occurs by incrementing or decrementing permanence values.

We denote the synaptic permanences of the dth dendritic segment of the ith input cell in the jth mini-column as a $N^{ext} \times 1$ vector $\mathbf{D}^{ijd,in}$. Similarly, the permanences of the dth dendritic segment of the ith output cell is the $N^{out}N^c \times 1$ dimensional vector $\mathbf{D}^{id,out}$.

Output neurons receive feedforward connections from input neurons within the same cortical column. We denote these connections with a $M \times N^{in} \times N^{out}$ tensor $\mathbf{F}$, where f_{ijk} represents the permanence of the synapse between the ith input cell in the jth mini-column and the kth output cell.

For $\mathbf{D}$ and $\mathbf{F}$, we will use a dot (e.g., $\dot{\mathbf{D}}$) to denote the binary vector representing the subset of potential synapses on a segment (i.e., permanence value above 0). We use a tilde (e.g., $\tilde{\mathbf{D}}$) to denote the binary vector representing the subset of connected synapses (i.e., permanence value above connection threshold).

Initialization

Each dendritic segment is initialized to contain a random set of potential synapses. $\mathbf{D}^{ijd,in}$ is initialized to contain a random set of potential synapses chosen from the location input. Segments in $\mathbf{D}^{id,out}$ are initialized to contain a random set of potential synapses to other output cells. These can include cells from the same cortical column. We enforce the constraint that a given segment only contains synapses from a single column. In all cases the permanence values of potential synapses are chosen randomly: initially some are connected (above threshold) and some are unconnected.

Computing Cell States

A cell in the input layer is predicted if any of its basal distal segments have sufficient activity:

$$\pi_{ij}^{in} = \begin{cases} 1 \text{ if } \exists_d \left(\mathbf{L} \cdot \tilde{\mathbf{D}}^{ijd,in} \geq \theta_b^{in} \right) > 0 \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

where θ_b^{in} is the activation threshold of the basal distal dendrite of an input cell.

For the input layer, all the cells in a mini-column share the same feedforward receptive fields. Following (Hawkins and Ahmad, 2016) we assume that an inhibitory process selects a set of s mini-columns that best match the current feedforward input pattern. We denote this winner set as $\mathbf{W}^{in}$. The set of active input layer cells is calculated as follows:

$$a_{ij}^{in} = \begin{cases} 1 \text{ if } j \in \mathbf{W}^{in} \text{ and } \pi_{ij}^{in} > 0 \\ 1 \text{ if } j \in \mathbf{W}^{in} \text{ and } \sum_i \pi_{ij}^{in} = 0 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

The first conditional states that predicted cells in a winning mini-column becoming winners and become active. If no cell in a mini-column is predicted, all cells in that mini-column become active (second conditional).

To determine activity in the output layer we calculate the feedforward and lateral input to each cell. Cells with enough feedforward overlap with the input layer, and the most lateral support from the previous time step become active. The feedforward overlap to the kth output cell is:

$$o_k^{out,\,t} = \sum_{i,j} I[f_{ijk} \geq \theta_c^{out}] a_{ij}^{in,t} \tag{3}$$

The set of output cells with enough feedforward input is computed as:

$$\mathbf{W}^{out,\,t} = \left\{ k | o_k^{out,\,t} \geq \theta_p^{out} \right\} \tag{4}$$

where θ_p^{out} is a threshold. We then select the active cells using the number of active basal segments as a sorting function:

$$a_i^{out,\,t} = \begin{cases} 1 \text{ if } i \in \mathbf{W}^{out,t} \text{ and } \rho_i^{out,t-1} \geq \xi_{t-1}^{out} \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

where $\rho_i^{out,t-1} = \sum_d I[\tilde{\mathbf{A}}^{out} \cdot \tilde{\mathbf{D}}^{id,out} \geq \theta_b^{out}]$ represents the number of active basal segments in the previous time step, and the sth highest number of active basal segments is denoted as ξ_t^{out}. θ_b^{out} is the activation threshold of the basal distal dendrite of an output cell. $I[]$ is the indicator function, and s is the minimum desired number of active neurons. If the number of cells with lateral support is less than s in a cortical column, $\xi_t^{out,\,c}$ would be zero and all cells with enough feedforward input will become active. Note that we used a modified version of the original HTM neuron model in the output layer by considering the effect of multiple active basal segments.

Learning in the Input Layer

In the input layer, basal segments represent predictions. At any point only segments that match its contextual input are modified. If a cell was predicted (Equation 1) and becomes active, the corresponding basal segments are selected for learning. If no cell in an active mini-column was predicted, we select a winning cell as the cell with the best basal input match via random initial conditions.

For selected segments, we decrease the permanence of inactive synapses by a small value p^- and increase the permanence of active synapses by a larger value p^+

$$\Delta \mathbf{D}^{ijd,in} = p_{in}^+ \dot{\mathbf{D}}^{ijd,in} \circ \mathbf{L}^t - p_{in}^- \dot{\mathbf{D}}^{ijd,in} \circ (1 - \mathbf{L}^t) \tag{6}$$

where $\circ$ represents element-wise multiplication. Incorrect predictions are negatively punished. If a basal dendritic segment on a cell becomes active and the cell subsequently does not become active, we slightly decrement the permanences of active synapses on the corresponding segments. Note that in Equation (6), learning is applied to all potential synapses (denoted by $\dot{\mathbf{D}}$).

Learning in the Output Layer

When learning a new object a sparse set of cells in the output layer is selected to represent the new object. These cells remain active while the system senses the object at different locations. Thus, each output cell pools over multiple feature/location representations in the input layer.

For each sensation, proximal synapses are learned by increasing the permanence of active synapses by p_{ff}^+, and decreasing the permanence of inactive synapses by p_{ff}^-:

$$\Delta f_{ijk} = [p_{ff}^+ a_{ij}^{in} - p_{ff}^- (1 - a_{ij}^{in})] I[f_{ijk} > 0] \tag{7}$$

Basal segments of active output cells are learned using a rule similar to Equation (7):

$$\Delta \mathbf{D}^{id,out} = p_{out}^+ \dot{\mathbf{D}}^{id,out} \circ \overline{\mathbf{A}}^{out,t-1} - p_{out}^- \dot{\mathbf{D}}^{id,out} \circ (1 - \overline{\mathbf{A}}^{out,t-1}) \tag{8}$$

Feedback

Feedback from the output layer to the input layer is used as an additional modulatory input to fine tune which cells in a winning mini-column become active. Cells in the input layer maintain a set of apical segments similar to the set of basal segments. If a cell has apical support (i.e., an active apical segment), we use a slightly lower value of θ_b^{in} to calculate π_{ij}^{in}. In addition if multiple cells in a mini-column are predicted, only cells with feedback become active. These rules make the set of active cells more precise with respect to the current representation in the output layer. Apical segments on winning cells in the input layer are learned using exactly the same rules as basal segments.

Simulation Details

To generate our convergence and capacity results we generated a large number of objects. Each object consists of a number of sensory features, with each feature assigned to a corresponding location. We encode each location as a 2,400-dimensional sparse binary vector with 10 random bits active. Each sensory feature is similarly encoded by a vector with 10 random bits active. The length of the sensory feature vector is the same as the number of mini-columns of the input layer N^{in}. The input layer contains 150 mini-columns and 16 cells per mini-column, with 10 mini-columns active at any time. The activation threshold of basal distal dendrite of input neuron is 6. The output layer contains 4,096 cells and the minimum number of active output cells is 40. The activation threshold is 3 for proximal dendrites and 18 for basal dendrites for output neurons.

During training, the network learns each object in random order. For each object, the network senses each feature three times. The activation pattern in the output layer is saved for each object to calculate retrieval accuracy. During testing, we allow the network to sense each object at K locations. After each sensation, we classify the activity pattern in the output layer. We say that an object is correctly classified if, for each cortical column, the overlap between the output layer and the stored representation for the correct object is above a threshold, and the overlaps with the stored representation for all other objects are below that threshold. We use a threshold of 30.

For the network convergence experiment (**Figures 4**, **5**), each object consists of 10 sensory features chosen from a library of 5 to 30 possible features. The number of sensations during testing is 20. For the capacity experiment, each object consists of 10 sensory features chosen from a large library of 5,000 possible features. The number of sensations during testing is 3.

Finally, we make some simplifying assumptions that greatly speed up simulation time for larger networks. Instead of explicitly initializing a complete set of synapses across every segment and every cell, we greedily create segments on a random cell and initialize potential synapses on that segment by sampling from currently active cells. This happens only when there is no match to any existing segment.

For the noise robustness experiment (**Figure 6**), we added random noise to the sensory input and the location input. For each input, we randomly flip a fraction of the active input bits to inactive, and flip the corresponding number of inactive input bits to active. This procedure randomizes inputs while maintaining constant input sparsity. The noise level denotes the fraction of active input bits that are changed for each input. We varied the amount of noise between 0 and 0.7.

We constructed an ideal observer model to estimate the theoretical upper limit for model performance (**Figure 4C**, Supplementary Figure 9). During learning, the ideal observer model memorizes a list of (feature, location) pairs for each object. During inference, the ideal observer model stores the sequence of observed (feature, location) pairs and calculates the overlap between all the observed pairs and the memorized list of pairs for each object. The predicted object is the object that has the most overlap with all the observed sensations. To compare the ideal observer with a multi-column network with N columns, we provide it with N randomly chosen observations per sensation. Performance of the ideal observer model represents the best one can do given all the sensations up to the current time. We also used the same framework to create a model that only uses sensory features, but no location signals (used in **Figure 4C**).

AUTHOR CONTRIBUTIONS

JH conceived of the overall theory and the detailed mapping to neuroscience, helped design the simulations, and wrote most of the paper. SA and YC designed and implemented the simulations and created the mathematical formulation of the algorithm.

FUNDING

Numenta is a privately held company. Its funding sources are independent investors and venture capitalists.

ACKNOWLEDGMENTS

We thank the reviewers for their detailed comments, which have helped to improve the paper significantly. We thank Jeff Gavornik for his thoughtful comments and suggestions. We also thank Marcus Lewis, Nathanael Romano, and numerous other collaborators at Numenta over the years for many discussions.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fncir.2017.00081/full#supplementary-material

REFERENCES

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-NC].

Ahveninen, J., Jääskeläinen, I. P., Raij, T., Bonmassar, G., Devore, S., Hämäläinen, M., et al. (2006). Task-modulated "what" and "where" pathways in human auditory cortex. *Proc. Natl. Acad. Sci. U.S.A.* 103, 14608–14613. doi: 10.1073/pnas.0510480103

Andersen, R. A., Snyder, L. H., Li, C. S., and Stricanne, B. (1993). Coordinate transformations in the representation of spatial information. *Curr. Opin. Neurobiol.* 3, 171–176. doi: 10.1016/0959-4388(93)90206-E

Bastos, A. M., Usrey, W. M., Adams, R. A., Mangun, G. R., Fries, P., and Friston, K. J. (2012). Canonical microcircuits for predictive coding. *Neuron* 76, 695–711. doi: 10.1016/j.neuron.2012.10.038

Billaudelle, S., and Ahmad, S. (2015). Porting htm models to the heidelberg neuromorphic computing platform. arXiv:1505.02142 [q-NC].

Binzegger, T., Douglas, R. J., and Martin, K. A. C. (2004). A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci.* 24, 8441–8453. doi: 10.1523/JNEUROSCI.1400-04.2004

Bolognini, N., and Maravita, A. (2007). Proprioceptive alignment of visual and somatosensory maps in the posterior parietal cortex. *Curr. Biol.* 17, 1890–1895. doi: 10.1016/j.cub.2007.09.057

Bremmer, F. (2000). Eye position effects in macaque area V4. *Neuroreport* 11, 1277–1283. doi: 10.1097/00001756-200004270-00027

Brotchie, P. R., Andersen, R. A., Snyder, L. H., and Goodman, S. J. (1995). Head position signals used by parietal neurons to encode locations of visual stimuli. *Nature* 375, 232–235. doi: 10.1038/375232a0

Brotchie, P. R., Lee, M. B., Chen, D. Y., Lourensz, M., Jackson, G., and Bradley, W. G. (2003). Head position modulates activity in the human parietal eye fields. *Neuroimage* 18, 178–184. doi: 10.1006/nimg.2002.1294

Bureau, I., Shepherd, G. M. G., and Svoboda, K. (2004). Precise development of functional and anatomical columns in the neocortex. *Neuron* 42, 789–801. doi: 10.1016/j.neuron.2004.05.002

Buxhoeveden, D. P. (2002). The minicolumn hypothesis in neuroscience. *Brain* 125, 935–951. doi: 10.1093/brain/awf110

Chapin, J. K. (1986). Laminar differences in sizes, shapes, and response profiles of cutaneous receptive fields in the rat SI cortex. *Exp. Brain Res.* 62, 549–559. doi: 10.1007/BF00236033

Chklovskii, D. B., Mel, B. W., and Svoboda, K. (2004). Cortical rewiring and information storage. *Nature* 431, 782–788. doi: 10.1038/nature03012

Constantinople, C. M., and Bruno, R. M. (2013). Deep cortical layers are activated directly by thalamus. *Science* 340, 1591–1594. doi: 10.1126/science.1236425

Craft, E., Schutze, H., Niebur, E., and von der Heydt, R. (2007). A neural model of figure-ground organization. *J. Neurophysiol.* 97, 4310–4326. doi: 10.1152/jn.00203.2007

DeSouza, J. F., Dukelow, S. P., and Vilis, T. (2002). Eye position signals modulate early dorsal and ventral visual areas. *Cereb. Cortex* 12, 991–997. doi: 10.1093/cercor/12.9.991

Douglas, R. J., and Martin, K. A. (1991). A functional microcircuit for cat visual cortex. *J. Physiol.* 440, 735–769. doi: 10.1113/jphysiol.1991.sp018733

Douglas, R. J., and Martin, K. A. (2004). Neuronal circuits of the neocortex. *Annu. Rev. Neurosci.* 27, 419–451. doi: 10.1146/annurev.neuro.27.070203.144152

Duhamel, J., Colby, C. L., and Goldberg, M. E. (1992). The updating of the representation of visual representation visual space in parietal cortex by intended eye movements. *Science* 255, 90–92. doi: 10.1126/science.1553535

Feldmeyer, D., Brecht, M., Helmchen, F., Petersen, C. C. H., Poulet, J. F. A., Staiger, J. F., et al. (2013). Barrel cortex function. *Prog. Neurobiol.* 103, 3–27. doi: 10.1016/j.pneurobio.2012.11.002

Feldmeyer, D., Lübke, J., Silver, R. A., and Sakmann, B. (2002). Synaptic connections between layer 4 spiny neurone-layer 2/3 pyramidal cell pairs in juvenile rat barrel cortex: physiology and anatomy of interlaminar signalling within a cortical column. *J. Physiol.* 538, 803–822. doi: 10.1113/jphysiol.2001.012959

Gavornik, J. P., and Bear, M. F. (2014). Learned spatiotemporal sequence recognition and prediction in primary visual cortex. *Nat. Neurosci.* 17, 732–737. doi: 10.1038/nn.3683

Gilbert, C. D. (1977). Laminar differences in receptive field properties of cells in cat primary visual cortex. *J. Physiol.* 268, 391–421. doi: 10.1113/jphysiol.1977.sp011863

Goodale, M. A., and Milner, A. D. (1992). Separate visual pathways for perception and action. *Trends Neurosci.* 15, 20–25. doi: 10.1016/0166-2236(92)90344-8

Graziano, M. S., and Gross, C. G. (1998). Spatial maps for the control of movement. *Curr. Opin. Neurobiol.* 8, 195–201. doi: 10.1016/S0959-4388(98)80140-2

Graziano, M. S., Hu, X. T., and Gross, C. G. (1997). Visuospatial properties of ventral premotor cortex. *J. Neurophysiol.* 77, 2268–2292.

Guillery, R. W., and Sherman, S. M. (2011). Branched thalamic afferents: what are the messages that they relay to the cortex? *Brain Res. Rev.* 66, 205–219. doi: 10.1016/j.brainresrev.2010.08.001

Guy, J., and Staiger, J. F. (2017). The functioning of a cortex without layers. *Front. Neuroanat.* 11:54. doi: 10.3389/fnana.2017.00054

Haeusler, S., and Maass, W. (2007). A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cereb. Cortex* 17, 149–162. doi: 10.1093/cercor/bhj132

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature* 436, 801–806. doi: 10.1038/nature03721

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

Helmstaedter, M., Sakmann, B., and Feldmeyer, D. (2009). Neuronal correlates of local, lateral, and translaminar inhibition with reference to cortical columns. *Cereb. Cortex* 19, 926–937. doi: 10.1093/cercor/bhn141

Hill, S., and Tononi, G. (2004). Modeling sleep and wakefulness in the thalamocortical system. *J. Neurophysiol.* 93, 1671–1698. doi: 10.1152/jn.00915.2004

Horton, J. C., and Adams, D. L. (2005). The cortical column: a structure without a function. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 360, 837–862. doi: 10.1098/rstb.2005.1623

Hubel, D., and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.* 160, 106–154. doi: 10.1113/jphysiol.1962.sp006837

Hunt, J. J., Bosking, W. H., and Goodhill, G. J. (2011). Statistical structure of lateral connections in the primary visual cortex. *Neural Syst. Circuits* 1:3. doi: 10.1186/2042-1001-1-3

Iyer, R., and Mihalas, S. (2017). Cortical circuits implement optimal context integration. *bioRxiv*. doi: 10.1101/158360

Jones, E. G. (2000). Microcolumns in the cerebral cortex. *Proc. Natl. Acad. Sci. U.S.A.* 97, 5019–5021. doi: 10.1073/pnas.97.10.5019

Kakei, S., Hoffman, D. S., and Strick, P. L. (2003). Sensorimotor transformations in cortical motor areas. *Neurosci. Res.* 46, 1–10. doi: 10.1016/S0168-0102(03)00031-2

Kim, E. J., Juavinett, A. L., Kyubwa, E. M., Jacobs, M. W., and Callaway, E. M. (2015). Three types of cortical layer 5 neurons that differ in brain-wide connectivity and function. *Neuron* 88, 1253–1267. doi: 10.1016/j.neuron.2015.11.002

Kim, J., Matney, C. J., Blankenship, A., Hestrin, S., and Brown, S. P. (2014). Layer 6 corticothalamic neurons activate a cortical output layer, layer 5a. *J. Neurosci.* 34, 9656–9664. doi: 10.1523/JNEUROSCI.1325-14.2014

Kropff, E., Carmichael, J. E., Moser, M.-B., and Moser, E. I. (2015). Speed cells in the medial entorhinal cortex. *Nature* 523, 419–424. doi: 10.1038/nature14622

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature* 521, 436–444. doi: 10.1038/nature14539

Lee, S., Carvell, G. E., and Simons, D. J. (2008). Motor modulation of afferent somatosensory circuits. *Nat. Neurosci.* 11, 1430–1438. doi: 10.1038/nn.2227

Lefort, S., Tomm, C., Floyd Sarria, J.-C., and Petersen, C. C. H. (2009). The excitatory neuronal network of the C2 barrel column in mouse primary somatosensory cortex. *Neuron* 61, 301–316. doi: 10.1016/j.neuron.2008.12.020

Lemmon, V., and Pearlman, A. L. (1981). Does laminar position determine the receptive field properties of cortical neurons? a study of corticotectal cells in area 17 of the normal mouse and the reeler mutant. *J. Neurosci.* 1, 83–93.

Li, N., and DiCarlo, J. J. (2008). Unsupervised natural experience rapidly alters invariant object representation in visual cortex. *Science* 321, 1502–1507. doi: 10.1126/science.1160028

Lohmann, H., and Rörig, B. (1994). Long-range horizontal connections between supragranular pyramidal cells in the extrastriate visual cortex of the rat. *J. Comp. Neurol.* 344, 543–558. doi: 10.1002/cne.903440405

Losonczy, A., Makara, J. K., and Magee, J. C. (2008). Compartmentalized dendritic plasticity and input feature storage in neurons. *Nature* 452, 436–441. doi: 10.1038/nature06725

Lubke, J., Egger, V., Sakmann, B., and Feldmeyer, D. (2000). Columnar organization of dendrites and axons of single and synaptically coupled excitatory spiny neurons in layer 4 of the rat barrel cortex. *J. Neurosci.* 20, 5300–5311.

Luhmann, H. J., Singer, W., and Martínez-Millán, L. (1990). Horizontal interactions in cat striate cortex: I. Anatomical substrate and postnatal development. *Eur. J. Neurosci.* 2, 344–357. doi: 10.1111/j.1460-9568.1990.tb00426.x

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Netw.* 10, 1659–1671. doi: 10.1016/S0893-6080(97)00011-7

Mangini, N. J., and Pearlman, A. L. (1980). Laminar distribution of receptive field properties in the primary visual cortex of the mouse. *J. Comp. Neurol.* 193, 203–222. doi: 10.1002/cne.901930114

Markov, N. T., Ercsey-Ravasz, M., Van Essen, D. C., Knoblauch, K., Toroczkai, Z., and Kennedy, H. (2013). Cortical high-density counterstream architectures. *Science* 342:1238406. doi: 10.1126/science.1238406

Markram, H., Muller, E., Ramaswamy, S., Reimann, M. W., Abdellah, M., Sanchez, C. A., et al. (2015). Reconstruction and simulation of neocortical microcircuitry. *Cell* 163, 456–492. doi: 10.1016/j.cell.2015.09.029

Martin, K. A. C., and Schröder, S. (2013). Functional heterogeneity in neighboring neurons of cat primary visual cortex in response to both artificial and natural stimuli. *J. Neurosci.* 33, 7325–7344. doi: 10.1523/JNEUROSCI.4071-12.2013

McGuire, B. A., Hornung, J. P., Gilbert, C. D., and Wiesel, T. N. (1984). Patterns of synaptic input to layer 4 of cat striate cortex. *J. Neurosci.* 4, 3021–3033.

Meyer, H. S., Schwarz, D., Wimmer, V. C., Schmitt, A. C., Kerr, J. N. D., Sakmann, B., et al. (2011). Inhibitory interneurons in a cortical column form hot zones of inhibition in layers 2 and 5A. *Proc. Natl. Acad. Sci. U.S.A.* 108, 16807–16812. doi: 10.1073/pnas.1113648108

Moser, E. I., Kropff, E., and Moser, M.-B. (2008). Place cells, grid cells, and the brain's spatial representation system. *Annu. Rev. Neurosci.* 31, 69–89. doi: 10.1146/annurev.neuro.31.061307.090723

Moser, E. I., Roudi, Y., Witter, M. P., Kentros, C., Bonhoeffer, T., and Moser, M.-B. (2014). Grid cells and cortical representation. *Nat. Rev. Neurosci.* 15, 466–481. doi: 10.1038/nrn3766

Moser, M.-B., Rowland, D. C., and Moser, E. I. (2015). Place cells, grid cells, and memory. *Cold Spring Harb. Perspect. Biol.* 7:a021808. doi: 10.1101/cshperspect.a021808

Mountcastle, V. (1978). "An organizing principle for cerebral function: the unit model and the distributed system," in *The Mindful Brain,* eds G. Edelman and V. Mountcastle (Cambridge, MA: MIT Press), 7–50.

Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain* 120, 701–722. doi: 10.1093/brain/120.4.701

Movshon, J. A., Thompson, I. D., and Tolhurst, D. J. (1978). Receptive field organization of complex cells in the cat's striate cortex. *J. Physiol.* 283, 79–99. doi: 10.1113/jphysiol.1978.sp012489

Nakamura, K., and Colby, C. L. (2002). Updating of the visual representation in monkey striate and extrastriate cortex during saccades. *Proc. Natl. Acad. Sci. U.S.A.* 99, 4026–4031. doi: 10.1073/pnas.052379899

Pouget, A., and Snyder, L. H. (2000). Computational approaches to sensorimotor transformations. *Nat. Neurosci.* 3, 1192–1198. doi: 10.1038/81469

Raizada, R. D. S., and Grossberg, S. (2003). Towards a theory of the laminar architecture of cerebral cortex: computational clues from the visual system. *Cereb. Cortex* 13, 100–113. doi: 10.1093/cercor/13.1.100

Ramaswamy, S., and Markram, H. (2015). Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron. *Front. Cell. Neurosci.* 9:233. doi: 10.3389/fncel.2015.00233

Reimann, M. W., Anastassiou, C. A., Perin, R., Hill, S. L., Markram, H., and Koch, C. (2013). A biophysically detailed model of neocortical local field potentials predicts the critical role of active membrane currents. *Neuron* 79, 375–390. doi: 10.1016/j.neuron.2013.05.023

Reyes-Puerta, V., Kim, S., Sun, J.-J., Imbrosci, B., Kilb, W., and Luhmann, H. J. (2015a). High stimulus-related information in barrel cortex inhibitory interneurons. *PLoS Comput. Biol.* 11:e1004121. doi: 10.1371/journal.pcbi.1004121

Reyes-Puerta, V., Sun, J.-J., Kim, S., Kilb, W., and Luhmann, H. J. (2015b). Laminar and columnar structure of sensory-evoked multineuronal spike sequences in adult rat barrel cortex *in vivo*. *Cereb. Cortex* 25, 2001–2021. doi: 10.1093/cercor/bhu007

Rizzolatti, G., Cattaneo, L., Fabbri-Destro, M., and Rozzi, S. (2014). Cortical mechanisms underlying the organization of goal-directed actions and mirror neuron-based action understanding. *Physiol. Rev.* 94, 655–706. doi: 10.1152/physrev.00009.2013

Rockland, K. S. (2010). Five points on columns. *Front. Neuroanat.* 4:22. doi: 10.3389/fnana.2010.00022

Russo, G. S., and Bruce, C. J. (1994). Frontal eye field activity preceding aurally guided saccades. *J. Neurophysiol.* 71, 1250–1253.

Rust, N. C., and DiCarlo, J. J. (2010). Selectivity and tolerance ("invariance") both increase as visual information propagates from cortical area V4 to IT. *J. Neurosci.* 30, 12978–12995. doi: 10.1523/JNEUROSCI.0179-10.2010

Sarid, L., Bruno, R., Sakmann, B., Segev, I., and Feldmeyer, D. (2007). Modeling a layer 4-to-layer 2/3 module of a single column in rat neocortex: interweaving *in vitro* and *in vivo* experimental observations. *Proc. Natl. Acad. Sci. U.S.A.* 104, 16353–16358. doi: 10.1073/pnas.0707853104

Schnepel, P., Kumar, A., Zohar, M., Aertsen, A., and Boucsein, C. (2015). Physiology and impact of horizontal connections in rat neocortex. *Cereb. Cortex* 25, 3818–3835. doi: 10.1093/cercor/bhu265

Sherman, S. M., and Guillery, R. W. (2011). Distinct functions for direct and transthalamic corticocortical connections. *J. Neurophysiol.* 106, 1068–1077. doi: 10.1152/jn.00429.2011

Shipp, S. (2007). Structure and function of the cerebral cortex. *Curr. Biol.* 17, R443–R449. doi: 10.1016/j.cub.2007.03.044

Spruston, N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nat. Rev. Neurosci.* 9, 206–221. doi: 10.1038/nrn2286

St. John-Saaltink, E., Kok, P., Lau, H. C., and de Lange, F. P. (2016). Serial dependence in perceptual decisions is reflected in activity patterns in primary visual cortex. *J. Neurosci.* 36, 6186–6192. doi: 10.1523/JNEUROSCI.4390-15.2016

Stettler, D. D., Das, A., Bennett, J., and Gilbert, C. D. (2002). Lateral connectivity and contextual interactions in macaque primary visual cortex. *Neuron* 36, 739–750. doi: 10.1016/S0896-6273(02)01029-2

Stuart, G. J., and Häusser, M. (2001). Dendritic coincidence detection of EPSPs and action potentials. *Nat. Neurosci.* 4, 63–71. doi: 10.1038/82910

Thomson, A. M. (2010). Neocortical layer 6, a review. *Front. Neuroanat.* 4:13. doi: 10.3389/fnana.2010.00013

Thomson, A. M., and Bannister, A. P. (2003). Interlaminar connections in the neocortex. *Cereb. Cortex* 13, 5–14. doi: 10.1093/cercor/13.1.5

Thomson, A. M., and Lamy, C. (2007). Functional maps of neocortical local circuitry. *Front. Neurosci.* 1, 19–42. doi: 10.3389/neuro.01.1.1.002.2007

Traub, R. D., Contreras, D., Cunningham, M. O., Murray, H., LeBeau, F. E. N., Roopun, A., et al. (2004). Single-column thalamocortical network model exhibiting gamma oscillations, sleep spindles, and epileptogenic bursts. *J. Neurophysiol.* 93, 2194–2232. doi: 10.1152/jn.00983.2004

Trotter, Y., and Celebrini, S. (1999). Gaze direction controls response gain in primary visual-cortex neurons. *Nature* 398, 239–242. doi: 10.1038/18444

Ungerleider, L. G., and Haxby, J. V. (1994). "What" and "where" in the human brain. *Curr. Opin. Neurobiol.* 4, 157–165. doi: 10.1016/0959-4388(94)90066-3

Viaene, A. N., Petrof, I., and Sherman, S. M. (2011a). Synaptic properties of thalamic input to layers 2/3 and 4 of primary somatosensory and auditory cortices. *J. Neurophysiol.* 105, 279–292. doi: 10.1152/jn.00747.2010

Viaene, A. N., Petrof, I., and Sherman, S. M. (2011b). Synaptic properties of thalamic input to the subgranular layers of primary somatosensory and auditory cortices in the mouse. *J. Neurosci.* 31, 12738–12747. doi: 10.1523/JNEUROSCI.1565-11.2011

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

Werner-Reiss, U., Kelly, K. A., Trause, A. S., Underhill, A. M., and Groh, J. M. (2003). Eye position affects activity in primary auditory cortex of primates. *Curr. Biol.* 13, 554–562. doi: 10.1016/S0960-9822(03)00168-4

Yen, S.-C., Baker, J., and Gray, C. M. (2006). Heterogeneity in the responses of adjacent neurons to natural stimuli in cat striate cortex. *J. Neurophysiol.* 97, 1326–1341. doi: 10.1152/jn.00747.2006

Zhou, H., Friedman, H. S., and von der Heydt, R. (2000). Coding of border ownership in monkey visual cortex. *J. Neurosci.* 20, 6594–6611.

Zipser, D., and Andersen, R. A. (1988). A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature* 331, 679–684. doi: 10.1038/331679a0

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest. JH, SA, and YC were employed by Numenta Inc. Numenta has some patents relevant to the work. Numenta has stated that use of its intellectual property, including all the ideas contained in this work, is free for non-commercial research purposes. In addition Numenta has released all pertinent source code as open source under a GPL V3 license (which includes a patent peace provision).

Unsupervised real-time anomaly detection for streaming data

Article *in* Neurocomputing · June 2017

DOI: 10.1016/j.neucom.2017.04.070

4 authors, including:

Subutai Ahmad
Numenta
53 PUBLICATIONS **1,399** CITATIONS
SEE PROFILE

Alexander Lavin
Numenta
7 PUBLICATIONS **196** CITATIONS
SEE PROFILE

Scott Purdy
Numenta
10 PUBLICATIONS **176** CITATIONS
SEE PROFILE

Neurocomputing 000 (2017) 1–14

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Unsupervised real-time anomaly detection for streaming data

Subutai Ahmad[a,*], Alexander Lavin[a], Scott Purdy[a], Zuha Agha[a,b]

[a] Numenta, Redwood City, CA, USA
[b] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

ARTICLE INFO

Article history:
Received 9 August 2016
Revised 19 April 2017
Accepted 22 April 2017
Available online xxx

Keywords:
Anomaly detection
Hierarchical Temporal Memory
Streaming data
Unsupervised learning
Concept drift
Benchmark dataset

ABSTRACT

We are seeing an enormous increase in the availability of streaming, time-series data. Largely driven by the rise of connected real-time data sources, this data presents technical challenges and opportunities. One fundamental capability for streaming analytics is to model each stream in an unsupervised fashion and detect unusual, anomalous behaviors in real-time. Early anomaly detection is valuable, yet it can be difficult to execute reliably in practice. Application constraints require systems to process data in real-time, not batches. Streaming data inherently exhibits concept drift, favoring algorithms that learn continuously. Furthermore, the massive number of independent streams in practice requires that anomaly detectors be fully automated. In this paper we propose a novel anomaly detection algorithm that meets these constraints. The technique is based on an online sequence memory algorithm called Hierarchical Temporal Memory (HTM). We also present results using the Numenta Anomaly Benchmark (NAB), a benchmark containing real-world data streams with labeled anomalies. The benchmark, the first of its kind, provides a controlled open-source environment for testing anomaly detection algorithms on streaming data. We present results and analysis for a wide range of algorithms on this benchmark, and discuss future challenges for the emerging field of streaming analytics.

1. Introduction

With sensors pervading our everyday lives, we are seeing an exponential increase in the availability of streaming, time-series data. Largely driven by the rise of the Internet of Things (IoT) and connected real-time data sources, we now have an enormous number of applications with sensors that produce important data that changes over time. Analyzing these streams effectively can provide valuable insights for any use case and application.

The detection of anomalies in real-time streaming data has practical and significant applications across many industries. Use cases such as preventative maintenance, fraud prevention, fault detection, and monitoring can be found throughout numerous industries such as finance, IT, security, medical, energy, e-commerce, agriculture, and social media. Detecting anomalies can give actionable information in critical scenarios, but reliable solutions do not yet exist. To this end, we propose a novel and robust solution to tackle the challenges presented by real-time anomaly detection.

Consistent with [1], we define an *anomaly* as a point in time where the behavior of the system is unusual and significantly different from previous, normal behavior. An anomaly may signify a negative change in the system, like a fluctuation in the turbine rotation frequency of a jet engine, possibly indicating an imminent failure. An anomaly can also be positive, like an abnormally high number of web clicks on a new product page, implying stronger than normal demand. Either way, anomalies in data identify abnormal behavior with potentially useful information. Anomalies can be *spatial*, where an individual data instance can be considered anomalous with respect to the rest of data, independent of where it occurs in the data stream, like the first and third anomalous spikes in Fig. 1. An anomaly can also be *temporal*, or *contextual*, if the temporal sequence of data is relevant; i.e., a data instance is anomalous only in a specific temporal context, but not otherwise. Temporal anomalies, such as the middle anomaly of Fig. 1, are often subtle and hard to detect in real data streams. Detecting temporal anomalies in practical applications is valuable as they can serve as an early warning for problems with the underlying system.

1.1. Streaming applications

Streaming applications impose unique constraints and challenges for machine learning models. These applications involve analyzing a continuous sequence of data occurring in real-time. In contrast to batch processing, the full dataset is not available. The

* Corresponding author.
E-mail address: sahmad@numenta.com (S. Ahmad).

http://dx.doi.org/10.1016/j.neucom.2017.04.070

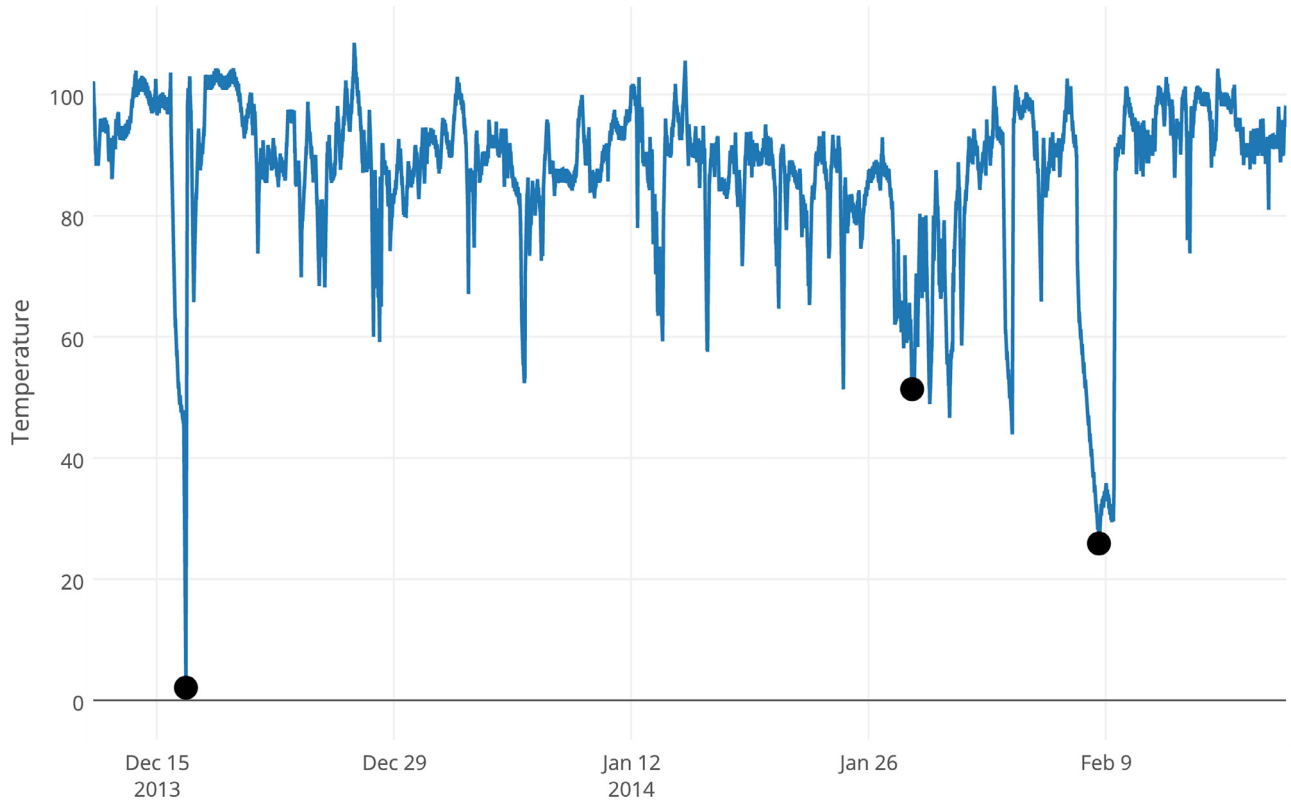


Fig. 1. The figure shows real-world temperature sensor data from an internal component of a large industrial machine. Anomalies are labeled with circles. The first anomaly was a planned shutdown. The third anomaly was a catastrophic system failure. The second anomaly, a subtle but observable change in the behavior, indicated the actual onset of the problem that led to the eventual system failure. The anomalies were hand-labeled by an engineer working on the machine. This file is included in the Numenta Anomaly Benchmark corpus [2].

system observes each data record in sequential order as they arrive and any processing or learning must be done in an online fashion. Let the vector $\boldsymbol{x}_t$ represent the state of a real-time system at time t. The model receives a continuous stream of inputs:

$$\ldots, \boldsymbol{x}_{t-2}, \boldsymbol{x}_{t-1}, \boldsymbol{x}_t, \boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+2}, \ldots$$

Consider for example, the task of monitoring a datacenter. Components of $\boldsymbol{x}_t$ might include CPU usage for various servers, bandwidth measurements, latency of servicing requests, etc. At each point in time t we would like to determine whether the behavior of the system is unusual. The determination must be made in real-time, before time $t+1$. That is, before seeing the next input ($\boldsymbol{x}_{t+1}$), the algorithm must consider the current and previous states to decide whether the system behavior is anomalous, as well as perform any model updates and retraining. Unlike batch processing, data is not split into train/test sets, and algorithms cannot look ahead.

Practical applications impose additional constraints on the problem. Typically, the sensor streams are large in number and at high velocity, leaving little opportunity for human, let alone expert, intervention; manual parameter tweaking and data labeling are not viable. Thus, operating in an unsupervised, automated fashion is often a necessity.

In many scenarios the statistics of the system can change over time, a problem known as *concept drift* [3,4]. Consider again the example of a production datacenter. Software upgrades and configuration changes can occur at any time and may alter the behavior of the system (Fig. 2). In such cases models must adapt to a new definition of "normal" in an unsupervised, automated fashion.

In streaming applications early detection of anomalies is valuable in almost any use case. Consider a system that continuously monitors the health of a cardiac patient's heart. An anomaly in the data stream could be a precursor to a heart attack. Detecting such an anomaly minutes in advance is far better than detecting it a few seconds ahead, or detecting it after the fact. Detection of anomalies often gives critical information, and we want this information early enough that it's actionable, possibly preventing system failure. There is a tradeoff between early detections and false positives, as an algorithm that makes frequent inaccurate detections is likely to be ignored.

Given the above requirements, we define the ideal characteristics of a real-world anomaly detection algorithm as follows:

1. Predictions must be made online; i.e., the algorithm must identify state $\boldsymbol{x}_t$ as normal or anomalous before receiving the subsequent $\boldsymbol{x}_{t+1}$.
2. The algorithm must learn continuously without a requirement to store the entire stream.
3. The algorithm must run in an unsupervised, automated fashion—i.e., without data labels or manual parameter tweaking.
4. Algorithms must adapt to dynamic environments and concept drift, as the underlying statistics of the data stream is often non-stationary.
5. Algorithms should make anomaly detections as early as possible.
6. Algorithms should minimize false positives and false negatives (this is true for batch scenarios as well).

Taken together, the above requirements suggest that anomaly detection for streaming applications is a fundamentally different problem than static batch anomaly detection. As discussed further below, the majority of existing anomaly detection algorithms

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

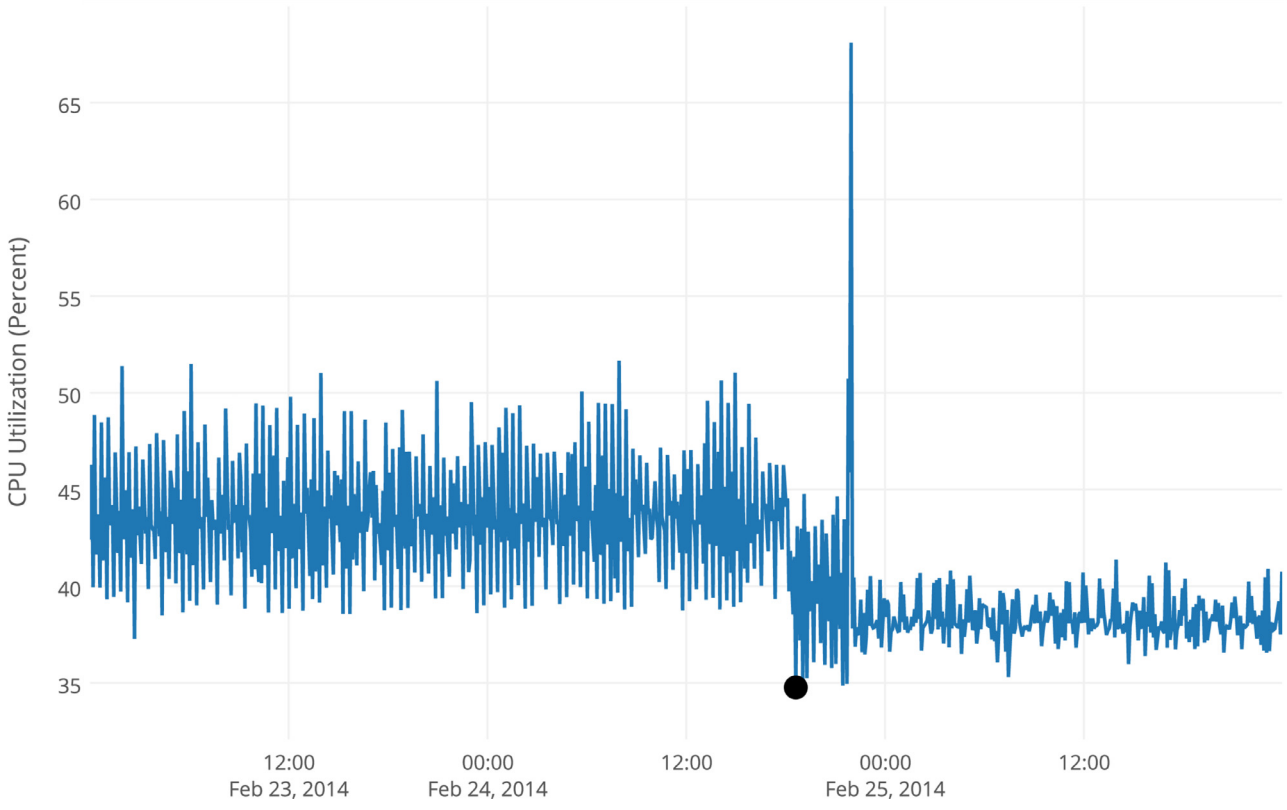


Fig. 2. CPU utilization (percent) for an Amazon EC2 instance (data from the Numenta Anomaly Benchmark [2]). A modification to the software running on the machine caused the CPU usage to change. The initial anomaly represents a changepoint, and the new system behavior that follows is an example of concept drift. Continuous learning is essential for performing anomaly detection on streaming data like this.

(even those designed for time-series data) are not applicable to streaming applications.

1.2. Related work

Anomaly detection in time-series is a heavily studied area of data science and machine learning, dating back to [5]. Many anomaly detection approaches exist, both supervised (e.g. support vector machines and decision trees [6]) and unsupervised (e.g. clustering), yet the vast majority of anomaly detection methods are for processing data in batches, and unsuitable for real-time streaming applications. Examples from industry include Netflix's robust principle component analysis (RPCA) method [7] and Yahoo's EGADS [8] both of which require analyzing the full dataset. Likewise, Symbolic Aggregate Approximation (SAX) [9] involves decomposing the full time series to generate symbols prior to anomaly detection. Other recent techniques include [10,11]. Although these techniques may work well in certain situations, they are traditional batch methods, and the focus of this paper is on methods for online anomaly detection. For reviews of anomaly detection in general we recommend [1,6,12,13]. For prior work on data stream mining and concept drift in general see [3,14–17].

Some anomaly detection algorithms are partially online. They either have an initial phase of offline learning, or rely on look-ahead to flag previously-seen anomalous data. Most clustering-based approaches fall under the umbrella of such algorithms. Some examples include Distributed Matching-based Grouping Algorithm (DMGA) [18], Online Novelty and Drift Detection Algorithm (OLINDDA) [19], and MultI-class learNing Algorithm for data Streams (MINAS) [20]. Another example is self-adaptive and dynamic k-means [21] that uses training data to learn weights prior to anomaly detection. Kernel-based recursive least squares (KRLS) proposed in [22] also violates the principle of no look-ahead as it resolves temporarily flagged data instances a few time steps later to decide if they were anomalous. However, some kernel methods, such as EXPoSE [23], adhere to our criteria of real-time anomaly detection (see evaluation section below).

For streaming anomaly detection, the majority of methods used in practice are statistical techniques that are computationally lightweight. These techniques include sliding thresholds, outlier tests such as extreme studentized deviate (ESD, also known as Grubbs') and k-sigma (e.g., [24,25]), changepoint detection [26], statistical hypotheses testing, and exponential smoothing such as Holt–Winters [27]. Typicality and eccentricity analysis [28,29] is an efficient technique that requires no user-defined parameters. Most of these techniques focus on spatial anomalies, limiting their usefulness in applications with temporal dependencies.

More advanced time-series modeling and forecasting models are capable of detecting temporal anomalies in complex scenarios. ARIMA is a general purpose technique for modeling temporal data with seasonality [30]. It is effective at detecting anomalies in data with regular daily or weekly patterns. Extensions of ARIMA enable the automatic determination of seasonality [31] for certain applications. A more recent example capable of handling temporal anomalies is the technique in [32] based on relative entropy.

Model-based approaches have been developed for specific use cases, but require explicit domain knowledge and are not generalizable. Domain-specific examples include anomaly detection in aircraft engine measurements [33], cloud datacenter temperatures [34], and ATM fraud detection [35]. Kalman filtering is a common technique, but the parameter tuning often requires domain knowledge and choosing specific residual error models [12,36–38]. Model-based approaches are often computationally efficient but their lack of generalizability limits their applicability to general streaming applications.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

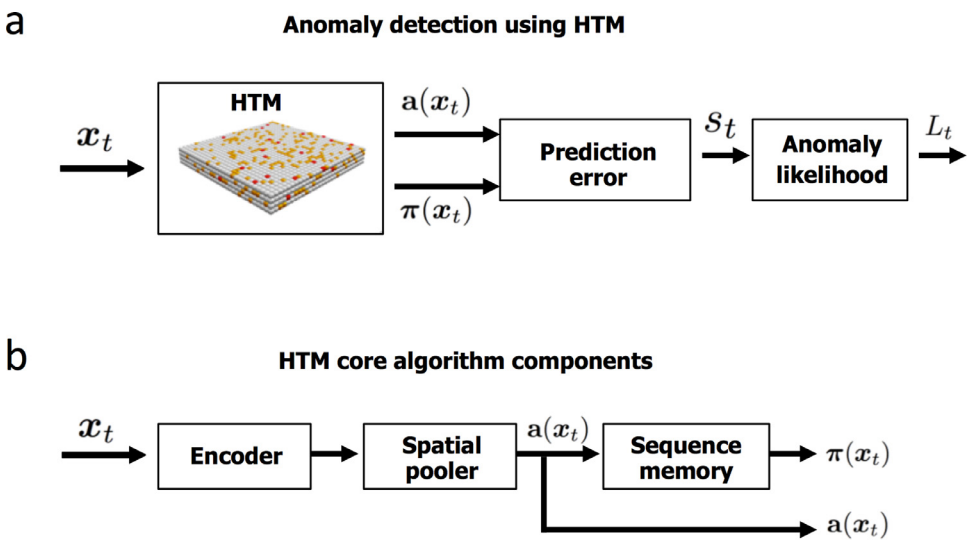


Fig. 3. (a) A block diagram outlining the primary functional steps used to create a complete anomaly detection system based on HTM. Our process takes the output of an HTM system and then performs two additional post-processing steps: computing the prediction error followed by computing an anomaly likelihood measure. (b) Breakdown of the core algorithm components within an HTM system.

There are a number of other restrictions that can make methods unsuitable for real-time streaming anomaly detection, such as computational constraints that impede scalability. An example is Lytics Anomalyzer [39], which runs in $O(n^2)$, limiting its usefulness in practice where streams are arbitrarily long. Dimensionality is another factor that can make some methods restrictive. For instance online variants of principle component analysis (PCA) such as osPCA [40] or window-based PCA [41] can only work with high-dimensional, multivariate data streams that can be projected onto a low dimensional space. Techniques that require data labels, such as supervised classification-based methods [42], are typically unsuitable for real-time anomaly detection and continuous learning.

Additional techniques for general purpose anomaly detection on streaming data include [9,43,44]. Twitter has an open-source method based on Seasonal Hybrid ESD [45]. Skyline is another popular open-source project, which uses an ensemble of statistical techniques for detecting anomalies in streaming data [24]. We include comparisons to both of these methods in our Results section.

1.3. Outline

The contributions of this paper are twofold: a novel anomaly detection technique built for real-time applications, and a comprehensive set of results on a benchmark designed for evaluating anomaly detection algorithms on streaming data. In Section 2 we show how to use Hierarchical Temporal Memory (HTM) networks [46–48] to robustly detect anomalies on a variety of data streams. The resulting system is efficient, extremely tolerant to noisy data, continuously adapts to changes in the statistics of the data, and detects subtle temporal anomalies while minimizing false positives. The HTM implementation and documentation are available as open-source.[1] In Section 3 we review the Numenta Anomaly Benchmark (NAB) [2], a rigorous benchmark dataset and scoring methodology we created for evaluating real-time anomaly detection algorithms. In Section 4 we present results comparing NAB on ten algorithms, many of which are commonly used in industry and academia. Section 5 concludes with a summary and directions for future work.

[1] HTM implementation is available at https://github.com/numenta/nupic.

2. Anomaly detection using HTM

Based on known properties of cortical neurons, Hierarchical Temporal Memory (HTM) is a theoretical framework for sequence learning in the cortex [46]. HTM implementations operate in real-time and have been shown to work well for prediction tasks [47,49]. HTM networks continuously learn and model the spatiotemporal characteristics of their inputs, but they do not directly model anomalies and do not output a usable anomaly score. In this section we describe our technique for applying HTM to anomaly detection.

Fig. 3(a) shows an overview of our process. At each point in time, the input data $\boldsymbol{x}_t$ is fed to a standard HTM network. We perform two additional computations on the output of the HTM. We first compute a measure of prediction error, s_t. Then, using a probabilistic model of s_t, we compute L_t, a likelihood that the system is in an anomalous state. A threshold on this likelihood determines whether an anomaly is detected. In the following subsections, we provide an overview of HTM systems and then describe our techniques for the additional steps of computing the prediction error and anomaly likelihood. Taken together, the algorithm fulfills the requirements for streaming applications outlined in Section 1.1.

2.1. Overview of HTM

Fig. 3(b) shows the core algorithm components and representations within a typical HTM system [49]. The current input, x_t, is fed to an encoder [50] and then a sparse spatial pooling process [51,52]. The resulting vector, $a(x_t)$, is a sparse binary vector representing the current input. The heart of the system is the sequence memory component. This component models temporal patterns in $a(x_t)$ and outputs a prediction in the form of another sparse vector $\pi(x_t)$. $\pi(x_t)$ is thus a prediction for $a(x_{t+1})$.

HTM sequence memory consists of a layer of HTM neurons organized into a set of columns (Fig. 4). The network accepts a stream of inputs encoded as sparse vectors. It models high-order sequences (sequences with long-term dependencies) using a composition of two separate sparse representations. The current input, $\boldsymbol{x}_t$ and the previous sequence context, $\ldots \boldsymbol{x}_{t-3}$, $\boldsymbol{x}_{t-2}$, $\boldsymbol{x}_{t-1}$, are simultaneously encoded using a dynamically updated sparse distributed representation. The network uses these representations to make predictions about the future in the form of a sparse vector.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

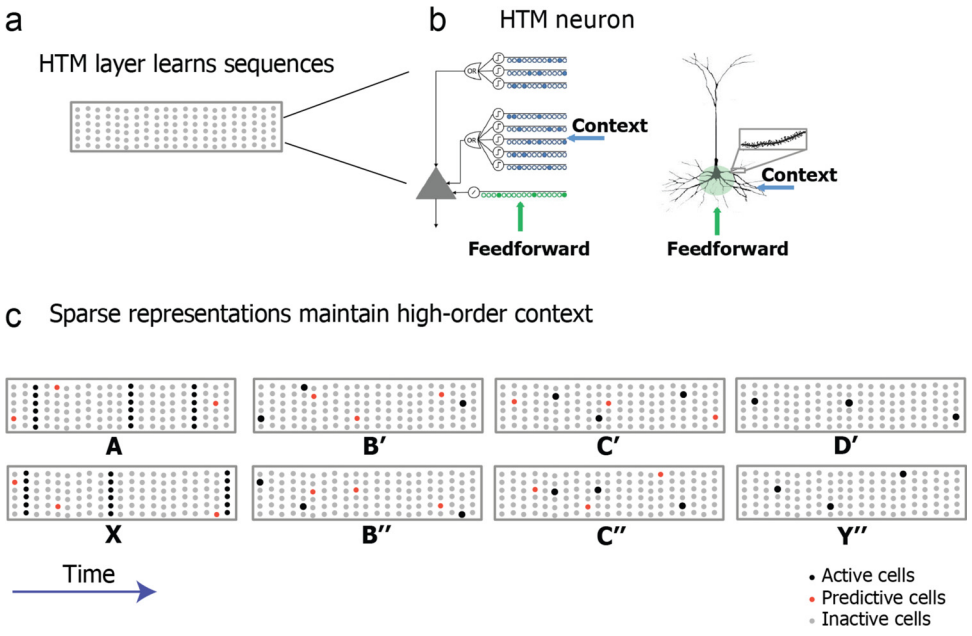


Fig. 4. The HTM sequence memory. (a) HTM sequence memory models one layer of cortex. The layer consists of a set of mini-columns, with each mini-column containing multiple neurons. (b) An HTM neuron (left) models the dendritic structure of pyramidal neurons in cortex (right). An HTM neuron models dendrites as an array of coincident detectors, each with a set of synapses. Context dendrites receive lateral input from other neurons within the layer. Each dendrite represents one transition in a sequence. Sufficient lateral activity on a context dendrite will cause the cell to enter a predicted state. (c) Representing high-order Markov sequences with shared subsequences (ABCD vs. XBCY). Each sequence element invokes a sparse set of cells within mini-columns. Cells that are predicted through lateral connections prevent other cells in the same column from firing through intra-column inhibition resulting in a highly sparse representation. As shown in the figure, such a representation can maintain past context. Because different cells respond to "C" in the two sequences (C' and C''), they can then invoke the correct prediction of either D or Y depending on the input from two time steps ago.

Fig. 4(c) shows how the sparse representations are used to represent temporal patterns and disambiguate sequences with long-term dependencies. When receiving the next input, the network uses the difference between predicted input and the actual input to update its synaptic connections. Learning happens at every time step but since the representations are highly sparse only a tiny percentage of the synapses are updated.

The details of the HTM learning algorithm and the properties of its representation are beyond the scope of this paper but are described in depth in [46,53]. In our implementation we use the standard HTM system [49] and a standard set of parameters (See Supplementary Section S3 for the complete list).

2.2. Computing the prediction error

Given the current input, $\boldsymbol{x}_t$, $\boldsymbol{a}(\boldsymbol{x}_t)$ is a sparse encoding of the current input, and $\boldsymbol{\pi}(\boldsymbol{x}_{t-1})$ is the sparse vector representing the HTM network's internal prediction of $\boldsymbol{a}(\boldsymbol{x}_t)$. The dimensionality of both vectors is equal to the number of columns in the HTM network (we use a standard value of 2048 for the number of columns in all our experiments). Let the *prediction error*, s_t, be a scalar value inversely proportional to the number of bits common between the actual and predicted binary vectors:

$$s_t = 1 - \frac{\boldsymbol{\pi}(\boldsymbol{x}_{t-1}) \cdot \boldsymbol{a}(\boldsymbol{x}_t)}{|\boldsymbol{a}(\boldsymbol{x}_t)|} \tag{1}$$

where $|\boldsymbol{a}(\boldsymbol{x}_t)|$ is the scalar norm, i.e. the total number of 1 bits in $\boldsymbol{a}(\boldsymbol{x}_t)$. In Eq. (1) the error s_t will be 0 if the current $\boldsymbol{a}(\boldsymbol{x}_t)$ perfectly matches the prediction, and 1 if the two binary vectors are orthogonal (i.e. they share no common 1 bits). s_t thus gives us an instantaneous measure of how well the underlying HTM model predicts the current input $\boldsymbol{x}_t$.

Changes to the underlying statistics are handled automatically due to the continuous learning nature of HTMs. If there is a shift in the behavior of the system, the prediction error will be high at the point of the shift, but will automatically degrade to zero as the model adapts to the "new normal". Fig. 5 shows an example stream and the behavior of the prediction error s_t. Shifts in the temporal characteristics of the system are handled in addition to spatial shifts in the underlying metric values.

An interesting aspect of this metric is that branching sequences are handled correctly. In HTMs, multiple predictions are represented in $\boldsymbol{\pi}(\boldsymbol{x}_t)$ as a binary union of each individual prediction. Similar to Bloom filters, as long as the vectors are sufficiently sparse and of sufficient dimensionality, a moderate number of predictions can be represented simultaneously with exponentially small chance of error [53,54]. The above error handles branching sequences gracefully in the following sense. If two completely different inputs are both possible and predicted, receiving either input will lead to a 0 error. Any other input will generate a positive error.

2.3. Computing anomaly likelihood

The prediction error described above represents an instantaneous measure of the predictability of the current input stream. As shown in Fig. 5, it works well for certain scenarios. In some applications however, the underlying system is inherently very noisy and unpredictable and instantaneous predictions are often incorrect. As an example, consider Fig. 6(a). This data shows the latency of a load balancer in serving HTTP requests on a production web site. Although the latency is generally low, it is not unusual to have occasional random jumps, leading to corresponding spikes in prediction error as shown in Fig. 6(b). The true anomaly is actually later in the stream, corresponding to a sustained increase in the frequency of high latency requests. Thresholding the prediction error directly would lead to many false positives.

To handle this class of scenarios, we introduce a second step. Rather than thresholding the prediction error s_t directly, we model

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

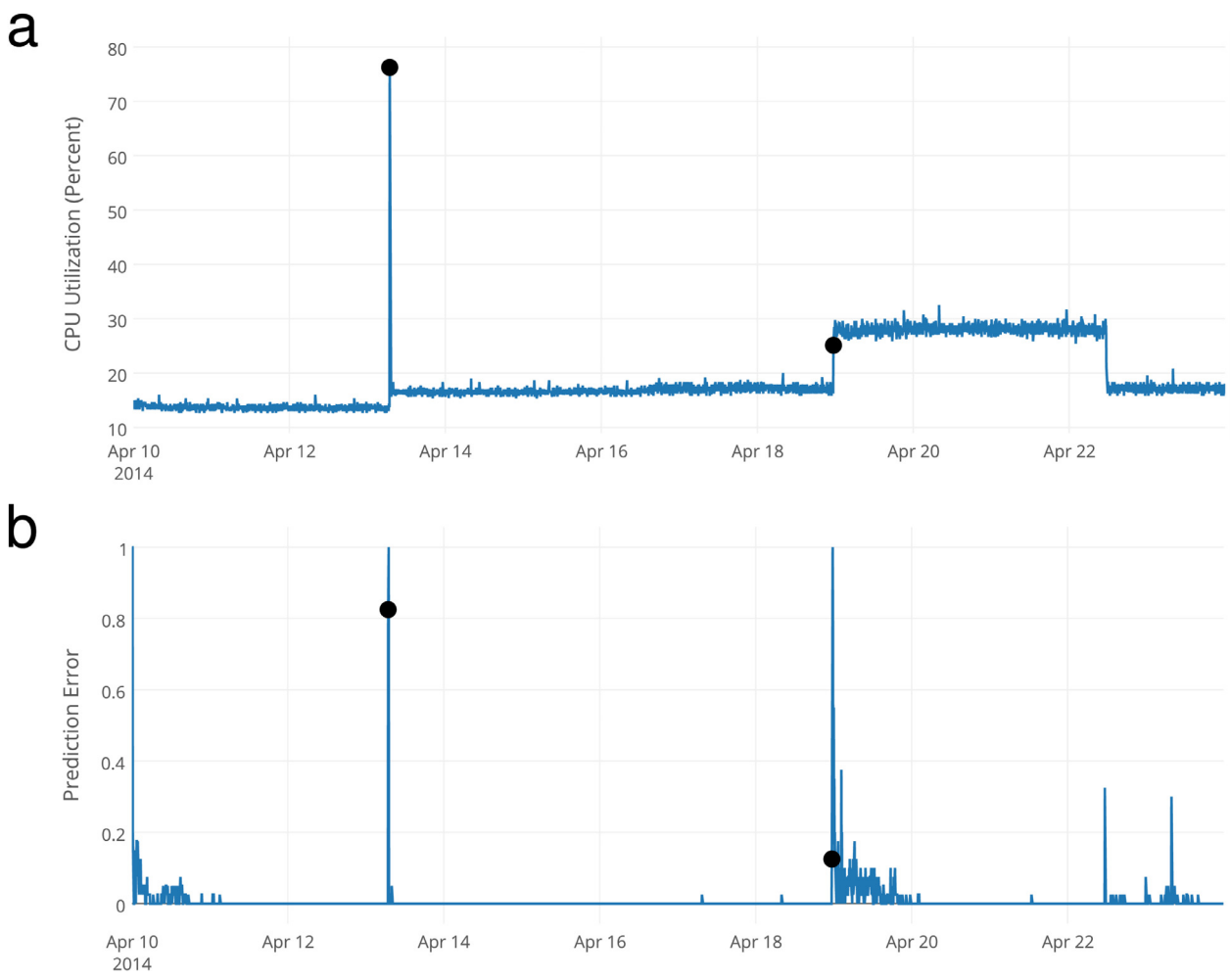


Fig. 5. An example stream along with prediction error. (a) This plot shows CPU usage on a database server over time. There are two unusual behaviors in this stream, the temporary spike up to 75% and the sustained shift up to 30% usage. B. This plot shows the prediction error while the HTM trains on this stream. Early during training the prediction error is high while the HTM model learns the data. There is a spike in prediction error corresponding to the temporary spike in CPU usage that quickly drops once usage goes back near normal. Finally there is an increase in prediction error corresponding to the sustained shift, which drops after the HTM has learned the new behavior.

the distribution of error values as an indirect metric, and use this distribution to check for the likelihood that the current state is anomalous. The *anomaly likelihood* is thus a probabilistic metric defining how anomalous the current state is based on the prediction history of the HTM model. To compute the anomaly likelihood we maintain a window of the last W error values. We model the distribution as a rolling normal distribution[2] where the sample mean, μ_t, and variance, σ_t^2, are continuously updated from previous error values as follows:

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W} \tag{2}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{W-1} \tag{3}$$

We then compute a recent short term average of prediction errors, and apply a threshold to the Gaussian tail probability (Q-function [55]) to decide whether or not to declare an anomaly.[3] We define the *anomaly likelihood* as the complement of the tail probability:

$$L_t = 1 - Q\left(\frac{\tilde{\mu}_t - \mu_t}{\sigma_t}\right) \tag{4}$$

where:

$$\tilde{\mu}_t = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{W'} \tag{5}$$

W' here is a window for a short term moving average, where $W' \ll W$, the duration for computing the distribution of prediction errors.[4] We threshold L_t based on a user-defined parameter ϵ to report an anomaly:

$$\text{anomaly detected}_t \equiv L_t \geq 1 - \epsilon \tag{6}$$

Since thresholding L_t involves thresholding a tail probability, there is an inherent upper limit on the number of alerts and a

[2] The distribution of prediction errors is not technically a normal distribution. We have also attempted to model the errors using a number of other distributions and distribution free bounds, such as Chebyshev's inequality. Modeling errors as a simple normal distribution worked significantly better than these other attempts. It is nevertheless still possible that modeling another distribution could improve results.

[3] The tail probability is the probability that a variable will be larger than x standard deviations above the mean.

[4] In practice we find that system performance is not sensitive to W as long as it is large enough to compute a reliable distribution. We use a generous value of $W = 8000$, and $W' = 10$.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

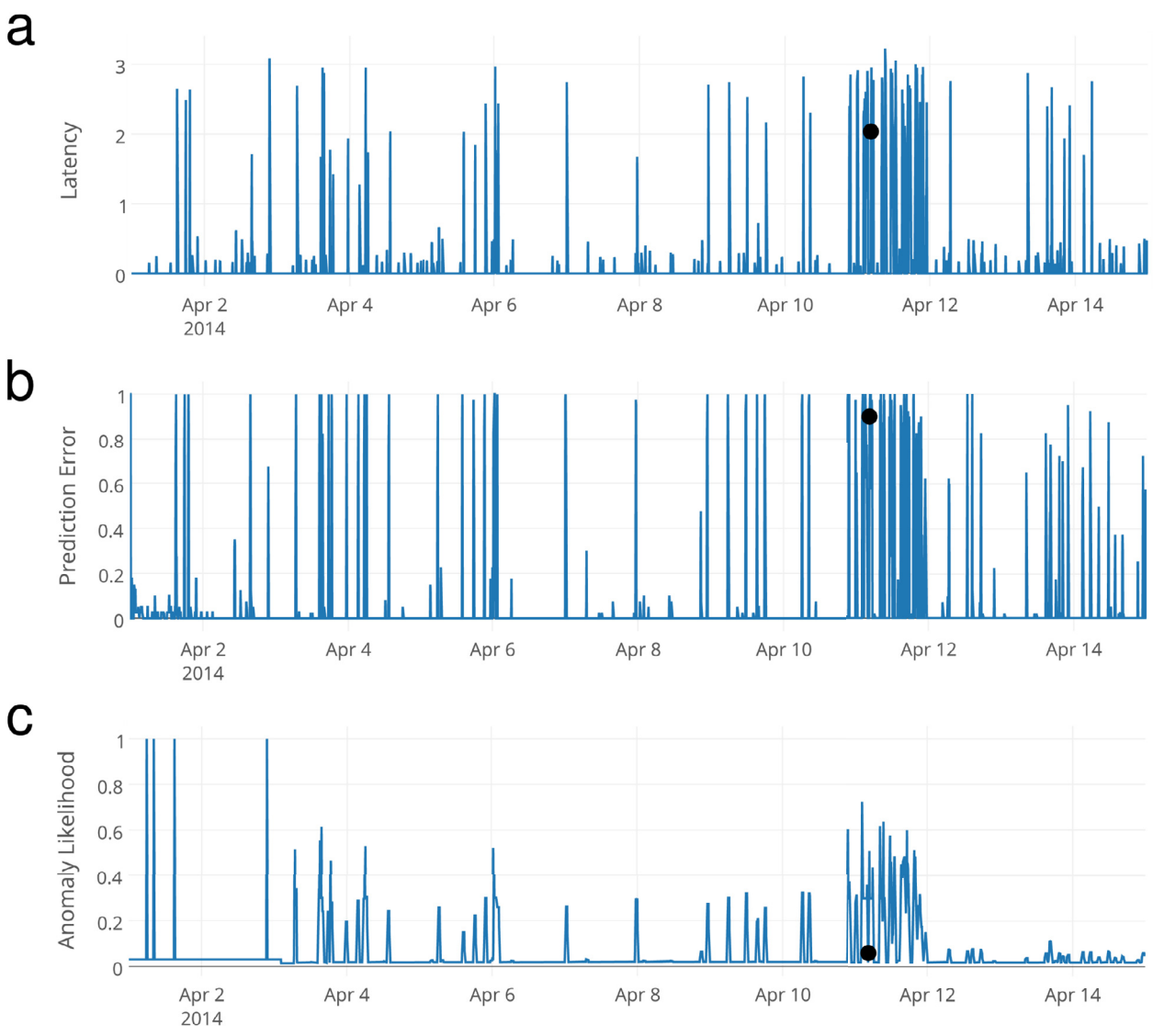


Fig. 6. A very noisy, unpredictable stream. (a) The data shows the latency (in seconds) of a load balancer on a production website. The anomaly (indicated by the dot) is an unusual sustained increase in latencies around April 10. (b) The prediction error from an HTM model on the latency values. The unpredictable nature of the latencies results in frequent spikes in the prediction error that cannot be distinguished from the true positives. The fact that the unpredictable metric values are spikes and the rest of the latencies are close to zero results in the coincidental similarity between the latencies and resulting prediction error. (c) A log-scale plot of the anomaly likelihood computed from the prediction error. Unlike the prediction error plot, there is a clear peak right around the real anomaly.

corresponding upper bound on the number of false positives. With ϵ very close to 0 it would be unlikely to get alerts with probability much higher than ϵ. In practice we have found that $\epsilon = 10^{-5}$ works well across a large range of domains and the user does not normally need to specify a domain-dependent threshold.

Fig. 6(c) shows an example of the anomaly likelihood, L_t, on noisy load balancer data. The figure demonstrates that the anomaly likelihood provides clearer peaks in extremely noisy scenarios compared to pure prediction error.

It is important to note that L_t is based on the distribution of prediction errors, not on the distribution of underlying metric values $\boldsymbol{x}_t$. As such, it is a measure of how well the model is able to predict, relative to the recent history. In clean predictable scenarios L_t behaves similarly to s_t. In these cases, the distribution of errors will have very small variance and will be centered near 0. Any spike in s_t will similarly lead to a corresponding spike in L_t. However, in scenarios with some inherent randomness or noise, the variance will be wider and the mean further from 0. A single spike in s_t will not lead to a significant increase in L_t but a series of spikes will. Importantly, a scenario that goes from wildly random to completely predictable will also trigger an anomaly.

2.4. Extensions

Modeling multiple streams simultaneously can enable the system to detect anomalies that cannot be detected from one stream alone. In Supplementary Section S4, we discuss this scenario and describe an extension for performing anomaly detection across multiple streams. We show how to combine independent models while accounting for temporal drift. This is particularly useful when there are many sensors and the combinations that enable detection are unknown.

Our algorithm is agnostic to the data type, as long as the data can be encoded as a sparse binary vector that captures the semantic characteristics of the data. We present an interesting extension with streaming geospatial data in Supplementary Section S2, demonstrating the applicability in diverse industries.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

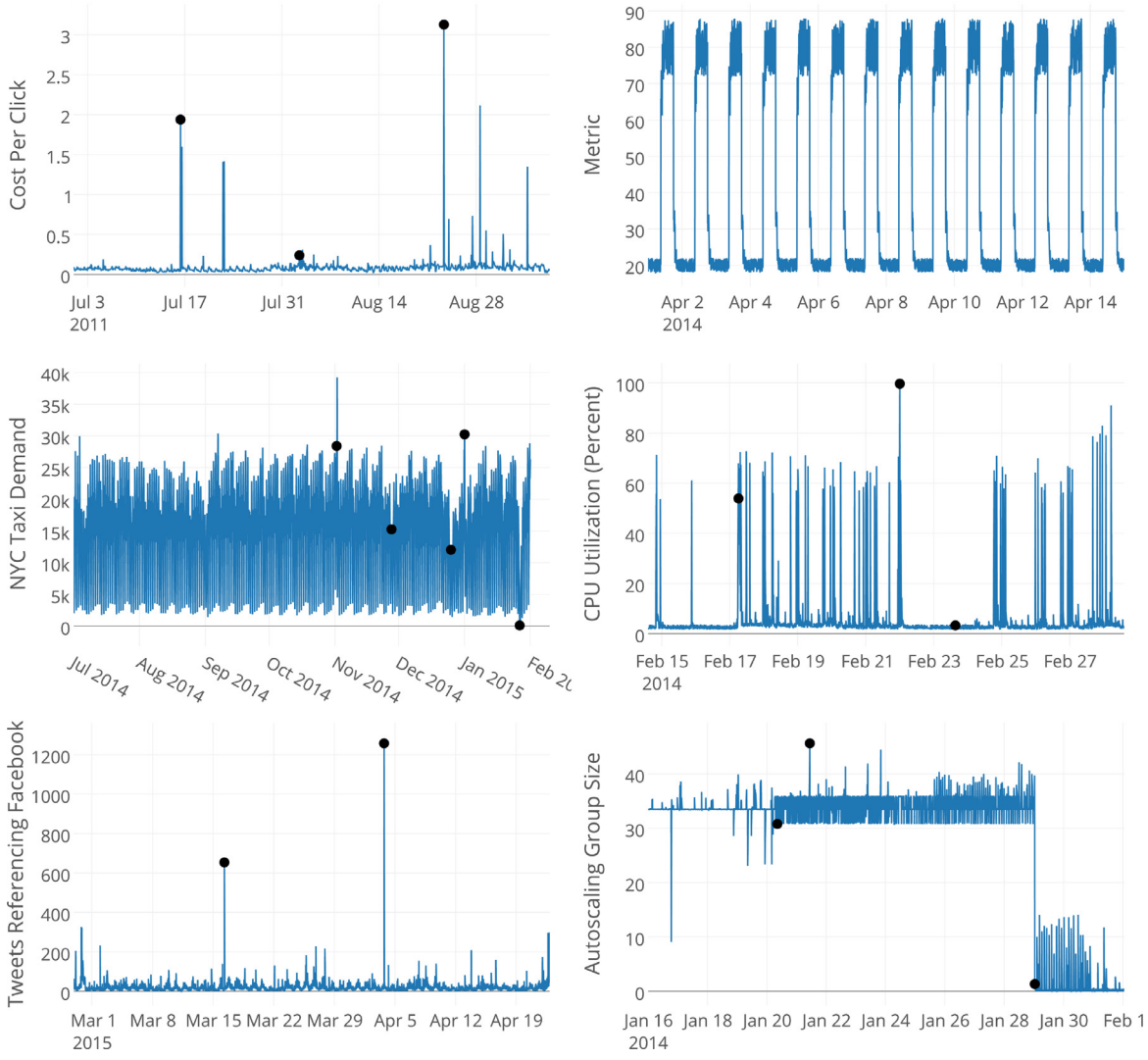


Fig. 7. Several data streams from the NAB corpus, showing a variety of data source and characteristics. From top left proceeding clockwise: click-through prices for online advertisements, an artificial stream with some noise but no anomalies, AWS Cloudwatch CPU utilization data, autoscaling group data for a server cluster, a stream of tweet volumes related to FB stock, and hourly demand for New York City taxis.

3. Evaluation of streaming anomaly detection algorithms

Numerous benchmarks exist for anomaly detection [1,56] but these benchmarks are generally designed for static datasets. Even benchmarks containing time-series data typically do not capture the requirements of real-time streaming applications. It is also difficult to find examples of real-world data that is labeled with anomalies. Yahoo released a dataset for anomaly detection in time-series data [8], but it is not available outside of academia and does not incorporate the requirements of streaming applications. As such we have created the Numenta Anomaly Benchmark (NAB) with the following goals:

1. Provide a dataset of labeled data streams from real-world streaming applications.
2. Provide a scoring methodology and set of constraints designed for streaming applications.
3. Provide a controlled open repository for researchers to evaluate and compare anomaly detection algorithms for streaming applications.

We briefly describe each of these below (full details can be found in [2]).

3.1. Benchmark dataset

The aim of the NAB dataset is to present algorithms with the challenges they will face in real-world scenarios, such as a mix of spatial and temporal anomalies, clean and noisy data, and data streams where the statistics evolve over time. The best way to do this is to provide data streams from real-world use cases, and from a variety of domains and applications. The data currently in the NAB corpus represents a variety of sources, ranging from server network utilization to temperature sensors on industrial machines to social media chatter.

NAB version 1.0 contains 58 data streams, each with 1000–22,000 records, for a total of 365,551 data points. Also included are some artificially-generated data files that test anomalous behaviors not yet represented in the corpus's real data, as well as several data files without any anomalies. All data files are labeled, either because we know the root cause for the anomalies from the provider of the data, or as a result of the well-defined NAB labeling procedure.[5] These labels define the ground truth anomalies used in the NAB scoring process. Fig. 1 is an example of noisy sensor data with spatial and temporal anomalies. Fig. 2 shows two related anomalies preceding a shift in the underlying statistics of the stream. Fig. 7 shows several data streams from the benchmark

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

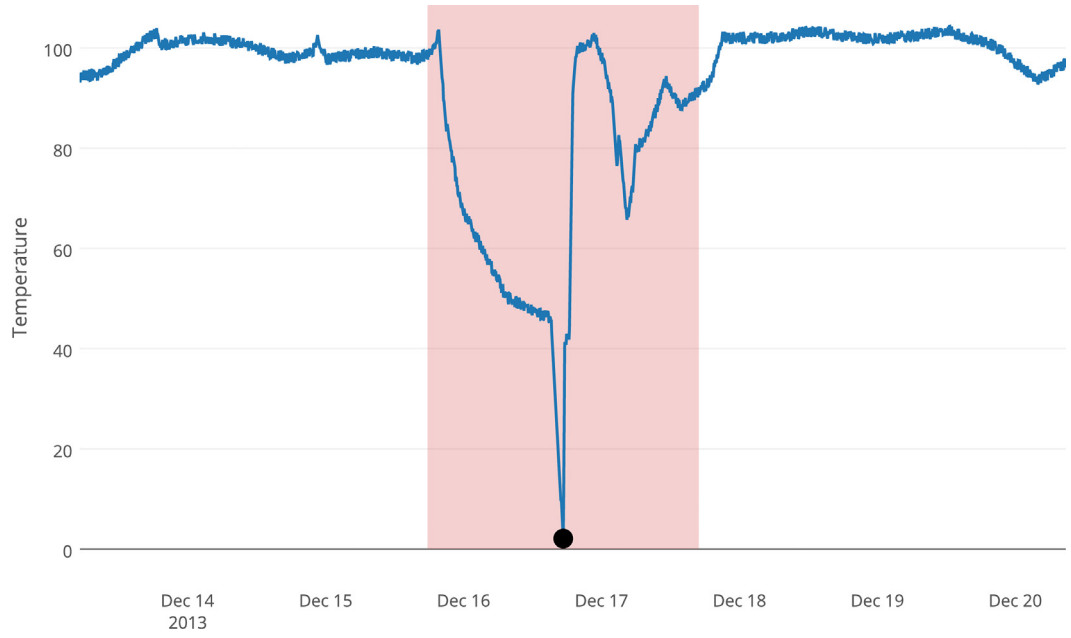


Fig. 8. A zoomed in view showing the anomaly window around the first anomaly in Fig. 1. The window size is set large enough to reward early detections of the anomaly, yet small enough such that poor detections count as false positives.

dataset, sourced from a variety of domains and exhibiting diverse characteristics such as temporal noise and short and long-term periodicities.

3.2. NAB scoring

The NAB scoring system formalizes a set of rules to determine the overall quality of streaming anomaly detection relative to the ideal, real-world anomaly detection algorithm that we defined earlier. There are three key aspects of scoring in NAB: anomaly windows, the scoring function, and application profiles. These are described below, and detailed discussion can be found in [2].

To incorporate the value of early detection into scoring, we define *anomaly windows* that label ranges of the data streams as anomalous, and a *scoring function* that uses these windows to reward early detections (and penalize later detections). When an algorithm is tested on NAB, the resulting detections must be scored. After an algorithm has processed a data file, the windows are used to identify true and false detections, and the scoring function is applied relative to each window to give value to the resulting true positives and false positives. Detections within a window correctly identify anomalous data and are true positives (TP), increasing the NAB score. If a detection occurs at the beginning of a window, it is given more value by the scoring function than a detection towards the end of the window. Multiple detections within a single window identify the same anomalous data, so we only use the earliest (most valuable) detection for the score contribution. Detections falling outside the windows are false positives, giving a negative contribution to the NAB score. The value of false positives is also calculated with the scoring function such that a false positive (FP) that occurs just after a window hurts the NAB score less than a FP that occurs far away from the window. Missing a window completely, or a false negative (FN), results in a negative contribution to the score. Refer to Supplementary Section S1 for details on scoring equations. All the scoring code and documentation is available in the repository.[5]

How large should the windows be? Large windows promote early detection of anomalies, but the tradeoff is that random or unreliable detections would be regularly reinforced. Using the underlying assumption that anomalous data is rare, the anomaly window length is defined to be 10%[6] the length of a data file, divided by the number of anomalies in the given file. This scheme provides a generous window to reward early detections and gives partial credit for detections slightly after the true anomaly, yet small enough such that poor detections are likely to be counted negatively (Fig. 8). Note that the streaming algorithms themselves have no information regarding the windows or the data length. Anomaly windows are only used as part of the benchmark and scoring system to gauge end performance.

NAB also includes a mechanism to evaluate algorithms on their bias towards false positives or false negatives. Depending on the application, a FN may be much more significant than a FP, such as detecting anomalies in ECG data, where a missed detection could be fatal. These scenarios are formalized in NAB by defining *application profiles* that vary the relative values of these metrics. For example, a datacenter application would be interested in the “Reward Low FP” profile, where false positives are weighted more heavily than in the other profiles.

The combination of anomaly windows, a smooth temporal scoring function, and application profiles allows researchers to evaluate online anomaly detector implementations against the requirements of the ideal detector. The NAB scoring system evaluates real-time performance, prefers earlier detection of anomalies, penalizes “spam” (i.e. FPs), and provides realistic costs for the standard classification evaluation metrics TP, FP, TN, and FN.

3.3. NAB is open-source

With the intent of fostering innovation in the field of anomaly detection, NAB is designed to be an accessible and reliable framework for all to use. Included with the open-source data and code is extensive documentation and examples on how to test algorithms.

[5] The NAB repository contains all the open-source data, code, and extensive documentation, including the labeling procedure and examples for running anomaly detection algorithms on NAB: https://github.com/numenta/NAB.

[6] We tested a range of window sizes (between 5% and 20%) and found that, partly due to the scaled scoring function, the end score was not sensitive to this percentage.

Table 1
NAB scoreboard showing results of each algorithm on v1.0 of NAB. Note the Random detector scores reflect the mean over a range of random seeds. HTM AL denotes the HTM algorithm using prediction error plus anomaly likelihood, as described in Section 2.2. HTM PE denotes the HTM algorithm using prediction error only. [+]Algorithms that were winners of the WCCI NAB competition.

Detector	Standard Profile	Reward Low FP	Reward Low FN
Perfect	100	100	100
HTM AL	70.1	63.1	74.3
CAD OSE[+]	69.9	67.0	73.2
nab-comportex[+]	64.6	58.8	69.6
KNN–CAD[+]	58.0	43.4	64.8
Relative Entropy	54.6	47.6	58.8
HTM PE	53.6	34.2	61.9
Twitter ADVec	47.1	33.6	53.5
Etsy Skyline	35.7	27.1	44.5
Sliding Threshold	30.7	12.1	38.3
Bayesian Changepoint	17.7	3.2	32.2
EXPoSE	16.4	3.2	26.9
Random	11	1.2	19.5
Null	0	0	0

Table 2
Comparison of properties for algorithms we implemented and tested on NAB (based on published information, excludes competition entries). Latency measures the time taken to process a single data point for anomaly detection. Latency time reported is an average over three runs on a single large data file from NAB, consisting of 22,695 data records. Timing was performed on an eight-core laptop with a 2.6 GHz Intel core i7 processor. NAB Score reflects the standard profile scores.

Detector	Latency (ms)	Spatial Anomaly	Temporal Anomaly	Concept Drift	Non Parametric	NAB Score
HTM	11.3	✔	✔	✔	✔	70.1
Relative Entropy	0.05	✔	✔	✔	✔	54.6
Twitter ADVec	3.0	✔	✔	✔	✗	47.1
Etsy Skyline	414.2	✔	✗	✗	✗	35.7
Sliding Threshold	0.4	✔	✗	✗	✗	30.7
Bayesian Changepoint	3.5	✔	✗	✔	✗	17.7
EXPoSE	2.6	✔	✔	✔	✔	16.4

The NAB repository contains source code for commonly used algorithms for online anomaly detection, as well as some algorithms submitted by the community.

4. Results & discussion

In this section we discuss NAB results and the comparative performance of a collection of real-time anomaly detection algorithms drawn from industry and academia. Our goals are to evaluate the performance of our HTM anomaly detection algorithm and, through a detailed discussion of the findings, to facilitate further research on unsupervised real-time anomaly detection algorithms.

4.1. Tested algorithms and parameter tuning

We considered a number of anomaly detection methods but the list was heavily filtered based on the criteria discussed in Sections 1.1 and 1.2. The algorithms evaluated include HTM, Twitter's Anomaly Detection, Etsy's Skyline, Multinomial Relative Entropy [32] EXPoSE [23], Bayesian Online Changepoint detection [57], and a simple sliding threshold. Some of these algorithms have open-source implementations and we implemented the rest based on their respective papers. We performed extensive parameter tuning for each algorithm; the resulting parameters are optimal to the best of our knowledge. Most of the algorithms also involve setting thresholds. The parameters were kept constant across all streams, and a single fixed threshold for each algorithm was used for the entire dataset, as required by NAB. Additional implementation details for these algorithms are included in Supplementary Section S5.

In addition, during the summer of 2016 we ran a NAB competition in collaboration with IEEE WCCI[7] to encourage additional algorithm testing. We include the result of the three competition winners below.

We use the HTM algorithm as described in Section 2. The core HTM algorithm by its nature is not highly sensitive to parameters. We used the architecture shown in Fig. 3 and the standard HTM parameter set (see Supplemental Section S3). The parameters specific to anomaly detection, ε, W, and W', were set to 10^{-5}, 8000, and 10, respectively. Timestamps and metric values in the NAB dataset were encoded using standard HTM datetime and scalar encoders [50]. In the results below, we show two variations. We show NAB results obtained by using prediction error only (i.e. thresholding s_t directly). We also show results obtained by using anomaly likelihood, as defined in Section 2.3.

For transparency and reproducibility, we have incorporated the source code and parameter settings for all of the above algorithms into the NAB repository.

4.2. Comparison of results

Table 1 summarizes the NAB scores for each algorithm across all application profiles including the three NAB competition winners. In addition to the algorithms described above, we also use three control detectors in NAB. A "null" detector runs through the dataset passively, making no detections, accumulating all false negatives. A "perfect" detector is an oracle that outputs detections that would maximize the NAB score; i.e., it outputs only true positives at the beginning of each window. The raw scores from these two detectors are used to scale the score for all other algorithms between 0 and 100. We also include a "random" detector that

[7] http://www.wcci2016.org.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

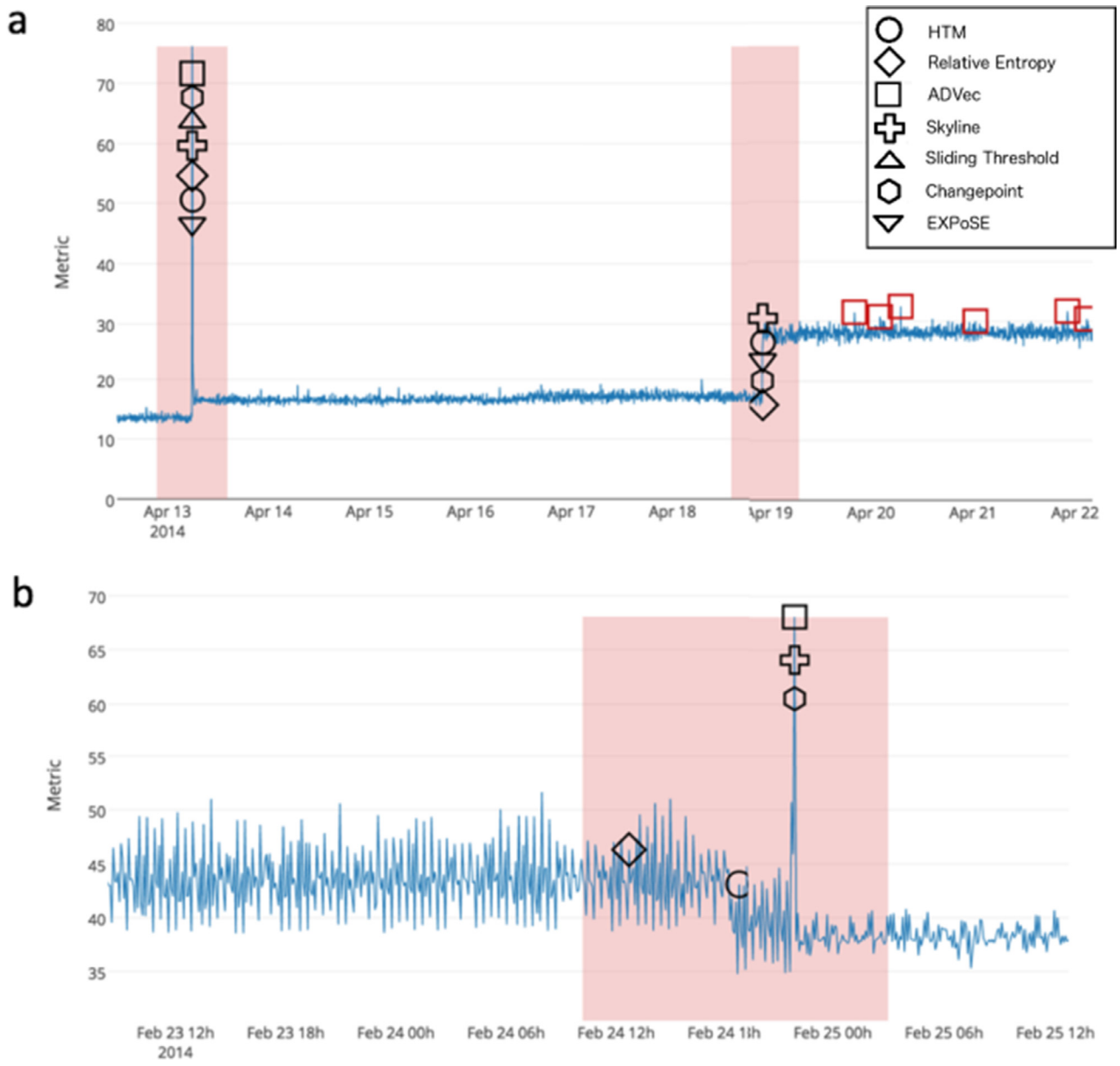


Fig. 9. These plots show detector results for two example NAB data streams. In both cases we show a subset of the full data stream. The plotted shapes correspond to the detections of seven different detectors: HTM, Multinomial Relative Entropy, Twitter ADVec, Skyline, Sliding Threshold, Bayesian Online Changepoint, and EXPoSE. Shapes that correspond to the same data point have been spaced vertically for clarity. For a given detector, true positive detections within each window (red shaded regions) are labeled in black. All false positive detections are colored red. (a) Detection results for a production server's CPU metric. The second anomaly shows a sustained shift that requires algorithms to adjust to a new normal behavior. (b) Results for the data stream shown in Fig. 2. Here we see a subtle temporal anomaly that preceded a large, obvious spike in the data. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

outputs a random anomaly probability for each data instance, which is then thresholded across the dataset for a range of random seeds. The score from this detector offers some intuition for chance-level performance on NAB.

Overall we see that the HTM detector using anomaly likelihood gets the best score, followed by CAD-OSE, nab-comportex, KNN–CAD, and the Multinomial Relative Entropy detector. The HTM detector using only prediction error performs moderately well, but using the anomaly likelihood step significantly improves the scores. Most of the algorithms perform much better than a random detector. There is a wide range of scores but none are close to perfect suggesting there is still significant room for improvement.

Table 2 summarizes the various algorithmic properties of each of the algorithms we implemented. Each algorithm is categorized based on its ability to detect spatial and temporal anomalies, handle concept drift, and automatically update parameters; these characteristics are based on published information, which may or may not reflect the actual performance. We also list the measured latency of processing each data point. Several algorithms claim to have all of the listed properties but their actual anomaly detection performance on the benchmark varies significantly. In general there is a rough correlation between the number of properties satisfied and the NAB ranking (with the exception of EXPoSE, see discussion below).

Based on a more detailed analysis of the results we highlight four factors that were important for achieving good performance on NAB: concept drift, ability to detect temporal anomalies, assumptions regarding distribution, and assumptions regarding the number of data points. We discuss each of these qualitatively below as well as more detailed file-level comparisons.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

Table 3
Comparison of average NAB scores for some algorithms across data from different kinds of sources provided by the benchmark. For each source, average scores are shown separately for data files that contain only spatial anomalies, only temporal anomalies and a mixture of spatial and temporal anomalies. The average score is given by $avg(a_s) = \frac{\sum_{i \in D(a_s)} score_i}{\sum_{i \in D(a_s)} counts_i}$ where $D(a_s)$ denotes data files from a given source s consisting of anomalies of the given type a, $score_i$ denotes the NAB score for the i^{th} file and $counts_i$ denotes the number of occurrences of the given anomaly type in i^{th} file. See Supplementary Section S6 for the scores and anomaly counts for each individual file. The best average scores across each row of algorithms are shown in **bold**. Note that scores are computed over standard profile with equal weights assigned to false positives and false negatives.

Source	Anomaly Type	Numenta HTM	CAD OSE	KNN CAD	Relative Entropy	Twitter AdVec	Skyline	Sliding Threshold
Artificial	Spatial only	0.70	**0.84**	−0.06	0.78	0.55	−0.39	−1.00
	Temporal only	0.11	0.08	−0.13	−0.52	**0.52**	−0.38	−0.90
	Spatial + Temporal	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Online advertisement clicks	Spatial only	**0.75**	0.21	0.54	−0.55	−0.03	−1.00	−0.38
	Temporal only	0.53	**0.83**	0.54	−0.53	−1.00	−1.00	−1.00
	Spatial + Temporal	**0.47**	**0.47**	0.37	−0.15	0.10	−0.47	0.41
AWS server metrics	Spatial only	0.61	**0.74**	0.54	0.11	0.28	0.40	−0.42
	Temporal only	**0.70**	0.29	0.03	0.53	−0.66	−0.77	−1.33
	Spatial + Temporal	**0.29**	0.20	0.01	−0.23	−0.45	−0.14	−0.34
Miscellaneous known causes	Spatial only	0.19	**0.33**	0.23	0.13	0.00	−0.38	−0.41
	Temporal only	−0.60	−0.79	**0.18**	−1.00	−0.91	−1.14	−0.96
	Spatial + Temporal	**0.32**	−0.15	−0.34	0.30	−0.48	−0.79	−0.92
Freeway traffic	Spatial only	0.83	0.85	−0.06	0.62	0.85	**0.87**	−0.38
	Temporal only	0.55	**0.86**	−1.44	0.75	−1.00	−1.00	−1.00
	Spatial + Temporal	0.51	**0.80**	0.26	0.50	−0.27	0.40	−0.83
Tweets volume	Spatial only	0.38	**0.74**	0.10	0.64	0.04	−1.46	−0.17
	Temporal only	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Spatial + Temporal	0.34	**0.43**	0.29	0.13	0.20	−0.25	0.05
Total Average		**0.40**	0.39	0.16	0.10	−0.03	−0.28	−0.36

The ability of each algorithm to learn continuously and handle concept drift is one of the contributing factors to obtaining a good NAB score. Fig. 9(a) demonstrates one example of this. This file shows CPU usage on a production server over time, and contains two anomalies. The first is a simple spike that is easily detectable by all algorithms. The second is a sustained shift in the usage, where the initial change in sequence is an anomalous changepoint, but the new behavior persists. Most algorithms detect the change and then adapt to the new normal. However, the Twitter ADVec algorithm fails to detect the changepoint and continues to generate anomalies for several days. The NAB corpus contains several similar examples, where an anomaly sets off a change in sequence that redefines the normal behavior. This is representative of one of the key challenges in streaming data. An inability to handle concept drift detections effectively results in a greater number of false positives, lowering the NAB score.

The ability to detect subtle temporal anomalies, while limiting false positives, is a second major factor in obtaining good NAB scores. In practical applications, one major benefit of detecting temporal patterns is that it often leads to the early detection of anomalies. Fig. 1 shows a representative example. The temporal anomaly in the middle of this figure precedes the actual failure by several days. In Fig. 2, the strong spike is preceded by a subtle temporal shift several hours earlier. Fig. 9(b) shows detection results on that data stream. The Twitter ADVec, Skyline, and Bayesian Online Changepoint algorithms easily detect the spike, but there are subtle changes in the data preceding the obvious anomaly. The Multinomial Relative Entropy and HTM detectors both flag anomalous data well before the large spike and thus obtain higher scores. It is challenging to detect such changes in noisy data without a large number of false positives. Both the HTM and Multinomial Relative Entropy perform well in this regard across the NAB dataset.

The third major factor concerns assumptions regarding the underlying distribution of data. A general lesson is that algorithms making fewer assumptions regarding distribution perform better. This is particularly important for streaming applications where algorithms must be unsupervised, automated, and applicable across a variety of domains. Techniques such as the sliding threshold and Bayesian Online Changepoint detectors make strong assumptions regarding the data and suffer as a result. Note that the Gaussian used in our anomaly likelihood technique is used to model the distribution of prediction errors, not the underlying metric data. As such it is a non-parametric technique with respect to the data.

Another interesting factor is demonstrated by the performance of EXPoSE. Theoretically EXPoSE possesses all the properties in Table 2, however it performs poorly on the benchmark. One of the reasons for this behavior is that EXPoSE has a dependence on the size of the dataset and is more suitable for large-scale datasets with high-dimensional features [58]. The technique computes an approximate mean kernel embedding and small or moderate data sets do not provide a sufficiently good proxy for this approximation. The average NAB data file contains 6300 records and is representative of real streaming applications. This issue highlights the need to output reliable anomalies relatively quickly.

4.3. Detailed NAB results

A breakdown of the algorithms performance on the benchmark is shown in Table 3. Results have been aggregated across data sources ranging from artificially generated streams to real streams from advertisement clicks, server metrics, traffic data and twitter volume. Data streams are characterized by spatial anomalies, temporal anomalies or a combination thereof. Grouping the streams by their anomaly types in Table 3 helps us inspect the characteristics of the algorithms identified earlier in Table 2.

Results show that both HTM and CAD-OSE yield the best overall aggregate scores on almost all data sources and anomaly types, with the exceptions of Twitter AdVec on artificial temporal streams and KNN–CAD on miscellaneous known causes. The difference between aggregate scores for HTM and CAD-OSE for the majority of the data streams is less than 0.20. For some stream types, HTM significantly outperforms CAD-OSE such as spatial advertisement streams, temporal server metric streams and spatial/temporal miscellaneous streams. In particular, the results show HTM performing well on server metrics and online advertisements data while CAD-OSE performs well on traffic and twitter streams.

In addition, the results in Table 3 also demonstrate that statistical techniques with assumptions on data distribution such as sliding threshold, Twitter AdVec and Skyline may be able to

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

capture spatial anomalies (e.g. Skyline on spatial traffic streams) but are not effective enough for capturing temporal anomalies. This is reflected by the negative scores for most temporal and spatial/temporal anomaly streams for these algorithms. This further reinforces the correlation between non-parametric techniques and detection of temporal anomalies.

5. Conclusion

With the increase in connected real-time sensors, the detection of anomalies in streaming data is becoming increasingly important. The use cases cut across a large number of industries. We believe anomaly detection represents one of the most significant near-term applications for machine learning in IoT.

In this paper we have discussed a set of requirements for unsupervised real-time anomaly detection on streaming data and proposed a novel anomaly detection algorithm for such applications. Based on HTM, the algorithm is capable of detecting spatial and temporal anomalies in predictable and noisy domains. The algorithm meets the requirements of real-time, continuous, online detection without look ahead and supervision.

We also reviewed NAB, an open benchmark for real-world streaming applications. We showed results of running a number of algorithms on this benchmark. We highlighted three key factors that impacted performance: concept drift, detection of temporal anomalies, and assumptions regarding distribution and size of data.

There are several areas for future work. The error analysis from NAB indicates that the errors across various algorithms (including HTM) are not always correlated. An ensemble-based approach might therefore provide a significant increase in accuracy. The current NAB benchmark is limited to data streams containing a single metric plus a timestamp. Adding real-world multivariate data streams labeled with anomalies, such as the data available in the DAMADICS dataset [59], would be a valuable addition.

Acknowledgments

We thank the anonymous reviewers for their feedback and helpful suggestions. We also thank Yuwei Cui, Jeff Hawkins, and Ian Danforth for many helpful comments, discussions, and suggestions.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.neucom.2017.04.070.

References

[1] V. Chandola, V. Mithal, V. Kumar, Comparative evaluation of anomaly detection techniques for sequence data, in: Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 743–748, doi:10.1109/ICDM.2008.151.
[2] A. Lavin, S. Ahmad, Evaluating real-time anomaly detection algorithms – the Numenta anomaly benchmark, in: Proceedings of the 14th International Conference on Machine Learning Application, Miami, Florida, IEEE, 2015, doi:10.1109/ICMLA.2015.141.
[3] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (2014) 1–37, doi:10.1145/2523813.
[4] M. Pratama, J. Lu, E. Lughofer, G. Zhang, S. Anavatti, Scaffolding type-2 classifier for incremental learning under concept drifts, Neurocomputing 191 (2016) 304–329, doi:10.1016/j.neucom.2016.01.049.
[5] A.J. Fox, Outliers in time series, J. R. Stat. Soc. Ser. B. 34 (1972) 350–363.
[6] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. 41 (2009) 1–72, doi:10.1145/1541880.1541882.
[7] Wong J. Netflix Surus GitHub, Online Code Repos https://github.com/Netflix/Surus 2015
[8] N. Laptev, S. Amizadeh, I. Flint, Generic and Scalable Framework for Automated Time-series Anomaly Detection, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery Data Mining, 2015, pp. 1939–1947.
[9] E. Keogh, J. Lin, A. Fu, HOT SAX: Efficiently finding the most unusual time series subsequence, in: Proceedings of the IEEE International Conference on Data Mining, ICDM, 2005, pp. 226–233, doi:10.1109/ICDM.2005.79.
[10] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long short term memory networks for anomaly detection in time series, Eur. Symp. Artif. Neural Netw. (2015) 22–24.
[11] H.N. Akouemo, R.J. Povinelli, Probabilistic anomaly detection in natural gas time series data, Int. J. Forecast. 32 (2015) 948–956, doi:10.1016/j.ijforecast.2015.06.001.
[12] J. Gama, Knowledge Discovery from Data Streams, Chapman and Hall/CRC, Boca Raton, Florida, 2010.
[13] M.A.F. Pimentel, D.A. Clifton, L. Clifton, L. Tarassenko, A review of novelty detection, Signal Process. 99 (2014) 215–249, doi:10.1016/j.sigpro.2013.12.026.
[14] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, Mining data streams, ACM SIGMOD Rec. 34 (2005) 18.
[15] M. Sayed-Mouchaweh, E. Lughofer, Learning in Non-Stationary Environments: Methods and Applications, Springer, New York, 2012.
[16] M. Pratama, J. Lu, E. Lughofer, G. Zhang, M.J. Er, Incremental learning of concept drift using evolving Type-2 recurrent fuzzy neural network, IEEE Trans. Fuzzy Syst (2016) 1, doi:10.1109/TFUZZ.2016.2599855.
[17] M. Pratama, S.G. Anavatti, M.J. Er, E.D. Lughofer, pClass: an effective classifier for streaming examples, IEEE Trans. Fuzzy Syst 23 (2015) 369–386, doi:10.1109/TFUZZ.2014.2312983.
[18] P.Y. Chen, S. Yang, J.A. McCann, Distributed real-time anomaly detection in networked industrial sensing systems, IEEE Trans. Ind. Electron 62 (2015) 3832–3842, doi:10.1109/TIE.2014.2350451.
[19] E.J. Spinosa, A.P.D.L.F. De Carvalho, J. Gama, OLINDDA: a cluster-based approach for detecting novelty and concept drift in data streams, in: Proceedings of the 2007 ACM Symposium on Applied Computing, 2007, pp. 448–452, doi:10.1145/1244002.1244107.
[20] E.R. Faria, J. Gama, A.C. Carvalho, Novelty detection algorithm for data streams multi-class problems, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 795–800, doi:10.1145/2480362.2480515.
[21] S. Lee, G. Kim, S. Kim, Self-adaptive and dynamic clustering for online anomaly detection, Expert Syst. Appl. 38 (2011) 14891–14898, doi:10.1016/j.eswa.2011.05.058.
[22] T. Ahmed, M. Coates, A. Lakhina, Multivariate online anomaly detection using kernel recursive least squares, in: Proceedings of the 26th IEEE International Conference on Computing Communication, 2007, pp. 625–633, doi:10.1109/INFCOM.2007.79.
[23] M. Schneider, W. Ertel, F. Ramos, Expected Similarity estimation for large-scale batch and streaming anomaly detection, Mach. Learn. 105 (2016) 305–333, doi:10.1007/s10994-016-5567-7.
[24] A. Stanway, Etsy Skyline, Online Code Repos. (2013). https://github.com/etsy/skyline.
[25] A. Bernieri, G. Betta, C. Liguori, On-line fault detection and diagnosis obtained by implementing neural algorithms on a digital signal processor, IEEE Trans. Instrum. Meas 45 (1996) 894–899, doi:10.1109/19.536707.
[26] M. Basseville, I. V Nikiforov, Detection of Abrupt Changes, 1993.
[27] M. Szmit, A. Szmit, Usage of modified holt-winters method in the anomaly detection of network traffic: case studies, J. Comput. Networks Commun. (2012), doi:10.1155/2012/192913.
[28] P. Angelov, Anomaly detection based on eccentricity analysis, in: Proceedings of the 2014 IEEE Symposium Evolving and Autonomous Learning Systems, 2014, doi:10.1109/EALS.2014.7009497.
[29] B.S.J. Costa, C.G. Bezerra, L.A. Guedes, P.P. Angelov, Online fault detection based on typicality and eccentricity data analytics, in: Proceedings of the International Joint Conference on Neural Networks, 2015, doi:10.1109/IJCNN.2015.7280712.
[30] A.M. Bianco, M. García Ben, E.J. Martínez, V.J. Yohai, Outlier detection in regression models with ARIMA errors using robust estimates, J. Forecast. 20 (2001) 565–579.
[31] R.J. Hyndman, Y. Khandakar, Automatic time series forecasting : the forecast package for R Automatic time series forecasting : the forecast package for R, J. Stat. Softw 27 (2008) 1–22.
[32] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, K. Schwan, Statistical techniques for online anomaly detection in data centers, in: Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management, 2011, pp. 385–392, doi:10.1109/INM.2011.5990537.
[33] D.L. Simon, A.W. Rinehart, A model-based anomaly detection approach for analyzing streaming aircraft engine measurement data, in: Proceedings of Turbo Expo 2014: Turbine Technical Conference and Exposition, ASME, 2014, pp. 665–672, doi:10.1115/GT2014-27172.
[34] E.K. Lee, H. Viswanathan, D. Pompili, Model-based thermal anomaly detection in cloud datacenters, in: Proceedings of the IEEE International Conference on Distributed Computing in Sensor Systems, 2013, pp. 191–198, doi:10.1109/DCOSS.2013.8.
[35] T. Klerx, M. Anderka, H.K. Buning, S. Priesterjahn, Model-based anomaly detection for discrete event systems, in: Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence, IEEE, 2014, pp. 665–672, doi:10.1109/ICTAI.2014.105.
[36] F. Knorn, D.J. Leith, Adaptive Kalman filtering for anomaly detection in software appliances, in: Proceedings of the IEEE INFOCOM, 2008, doi:10.1109/INFOCOM.2008.4544581.

[37] A. Soule, K. Salamatian, N. Taft, Combining filtering and statistical methods for anomaly detection, in: Proceedings of the 5th ACM SIGCOMM conference on Internet measurement, 4, 2005, p. 1, doi:10.1145/1330107.1330147.
[38] H. Lee, S.J. Roberts, On-line novelty detection using the Kalman filter and extreme value theory, in: Proceedings of the 19th International Conference on Pattern Recognition,, 2008, pp. 1–4, doi:10.1109/ICPR.2008.4761918.
[39] A. Morgan, Lytics Anomalyzer Blog, (2015). https://www.getlytics.com/blog/post/check_out_anomalyzer.
[40] Y.J. Lee, Y.R. Yeh, Y.C.F. Wang, Anomaly detection via online oversampling principal component analysis, IEEE Trans. Knowl. Data Eng 25 (2013) 1460–1470, doi:10.1109/TKDE.2012.99.
[41] A. Lakhina, M. Crovella, C. Diot, Diagnosing network-wide traffic anomalies, ACM SIGCOMM Comput. Commun. Rev 34 (2004) 219, doi:10.1145/1030194.1015492.
[42] N. Görnitz, M. Kloft, K. Rieck, U. Brefeld, Toward supervised anomaly detection, J. Artif. Intell. Res 46 (2013) 235–262, doi:10.1613/jair.3623.
[43] U. Rebbapragada, P. Protopapas, C.E. Brodley, C. Alcock, Finding anomalous periodic time series : An application to catalogs of periodic variable stars, Mach. Learn. 74 (2009) 281–313, doi:10.1007/s10994-008-5093-3.
[44] T. Pevný, Loda: Lightweight on-line detector of anomalies, Mach. Learn 102 (2016) 275–304, doi:10.1007/s10994-015-5521-0.
[45] A. Kejariwal, Twitter Engineering: Introducing Practical and Robust Anomaly Detection in a Time Series [Online blog], (2015). http://bit.ly/1xBbX0Z.
[46] J. Hawkins, S. Ahmad, Why neurons have thousands of synapses, a theory of sequence memory in neocortex, Front. Neural Circuits. 10 (2016) 1–13, doi:10.3389/fncir.2016.00023.
[47] D.E. Padilla, R. Brinkworth, M.D. McDonnell, Performance of a hierarchical temporal memory network in noisy sequence learning, in: Proceedings of the International Conference on Computational Intelligence and Cybernetics, IEEE, 2013, pp. 45–51, doi:10.1109/CyberneticsCom.2013.6865779.
[48] D. Rozado, F.B. Rodriguez, P. Varona, Extending the bioinspired hierarchical temporal memory paradigm for sign language recognition, Neurocomputing 79 (2012) 75–86, doi:10.1016/j.neucom.2011.10.005.
[49] Y. Cui, S. Ahmad, J. Hawkins, Continuous online sequence learning with an unsupervised neural network model, Neural Comput 28 (2016) 2474–2504, doi:10.1162/NECO_a_00893.
[50] S. Purdy, Encoding Data for HTM Systems, arXiv. (2016) arXiv:1602.05925 [cs.NE].
[51] J. Mnatzaganian, E. Fokoué, D. Kudithipudi, A Mathematical Formalization of hierarchical temporal memory's spatial pooler, Front. Robot. AI. 3 (2017) 81, doi:10.3389/frobt.2016.00081.
[52] Y. Cui, S. Ahmad, J. Hawkins, The HTM Spatial Pooler: a neocortical algorithm for online sparse distributed coding, bioRxiv, 2016, doi:http://dx.doi.org/10.1101/085035.
[53] S. Ahmad, J. Hawkins, Properties of sparse distributed representations and their application to Hierarchical Temporal Memory, 2015, arXiv:1503.07469 [q-NC].
[54] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM. 13 (1970) 422–426, doi:10.1145/362686.362692.
[55] G.K. Karagiannidis, A.S. Lioumpas, An improved approximation for the Gaussian Q-function, IEEE Commun. Lett 11 (2007) 644–646.
[56] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: A survey, ACM Comput. Surv (2009) 1–72.
[57] R.P. Adams, D.J.C. Mackay, Bayesian Online Changepoint Detection, 2007, arXiv:0710.3742 [stat.ML].
[58] M. Schneider, W. Ertel, G. Palm, Constant Time expected similarity estimation using stochastic optimization, (2015) arXiv:1511.05371 [cs.LG].
[59] M. Bartyś, R. Patton, M. Syfert, S. de las Heras, J. Quevedo, Introduction to the DAMADICS actuator FDI benchmark study, Control Eng. Pract. 14 (2006) 577–596, doi:10.1016/j.conengprac.2005.06.015.

Subutai Ahmad received his A.B. in Computer Science from Cornell University in 1986, and his PhD in Computer Science from the University of Illinois at Urbana-Champaign in 1991. He is the VP of Research at Numenta. His research interests are in computational neuroscience, machine learning, computer vision, and real-time systems.

Alexander Lavin received his B.S. in Mechanical Engineering from Cornell University in 2012, and M.S. in Mechanical Engineering from Carnegie Mellon University in 2014, specializing in spacecraft engineering. He is currently a Senior Research Engineer at Vicarious, building general artificial intelligence for robotics. His main research interests are computational neuroscience, computer vision systems, and robotics.

Scott Purdy received his B.S. and M.Eng. degrees in Computer Science in 2010 and 2011, respectively, from the College of Engineering at Cornell University. He is Director of Engineering at Numenta. Scott's research interests are computational neuroscience, machine learning, and robotics.

Zuha Agha received her B.S in Computer Science from Lahore University of Management Sciences Pakistan in 2014, and her M.S in Computer Science from University of Pittsburgh in 2017. She was formerly an Algorithms Intern at Numenta, and will join Apple in the summer of 2017. Her research interests include data science, machine learning, and computer vision.

Please cite this article as: S. Ahmad et al., Unsupervised real-time anomaly detection for streaming data, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.04.070

Locations in the Neocortex: A Theory of Sensorimotor Object Recognition Using Cortical Grid Cells

Preprint · October 2018

DOI: 10.1101/436352

4 authors, including:

Scott Purdy
Numenta
10 PUBLICATIONS **176** CITATIONS
SEE PROFILE

Subutai Ahmad
Numenta
53 PUBLICATIONS **1,399** CITATIONS
SEE PROFILE

Locations in the Neocortex: A Theory of Sensorimotor Object Recognition Using Cortical Grid Cells

Marcus Lewis, Scott Purdy, Subutai Ahmad, and Jeff Hawkins
Numenta, Inc., Redwood City, CA, USA

ABSTRACT

The neocortex is capable of modeling complex objects through sensorimotor interaction but the neural mechanisms are poorly understood. Grid cells in the entorhinal cortex represent the location of an animal in its environment, and this location is updated through movement and path integration. In this paper, we propose that grid-like cells in the neocortex represent the location of sensors on an object. We describe a two-layer neural network model that uses cortical grid cells and path integration to robustly learn and recognize objects through movement. Grid cells exhibit regular tiling over environments and are organized into modules, each with its own scale and orientation. A single module encodes position within the spatial scale of the module but is ambiguous over larger spaces. A set of modules can uniquely encode many large spaces. In our model, a layer of cells consisting of several grid-like modules represents a location in the reference frame of a specific object. Another layer of cells which processes sensory input receives this location input as context and uses it to encode the sensory input in the object's reference frame. Sensory input causes the network to invoke previously learned locations that are consistent with the input, and motor input causes the network to update those locations. Simulations show that the model can learn hundreds of objects even when object features alone are insufficient for disambiguation. We discuss the relationship of the model to cortical circuitry and suggest that the reciprocal connections between layers 4 and 6 fit the requirements of the model. We propose that the subgranular layers of cortical columns employ grid cell like mechanisms to represent object specific locations that are updated through movement.

INTRODUCTION

Our brains learn about the outside world by processing our sensory inputs and movements. As we touch an object, survey a visual scene, or explore an environment, the brain receives a series of sensations and movements, a *sensorimotor sequence*.

It's not clear how our brains extract reusable information from sensorimotor sequences. If the brain ignores the motor stream and learns objects using only the sensory stream, it will lose the ability to correctly combine information from multiple sensations. But if it learns objects by memorizing sensorimotor sequences, it won't be able to recognize familiar objects from novel sensorimotor sequences.

Most existing models of object recognition assume a strictly feedforward passive mechanism for object recognition (DiCarlo et al., 2012; Riesenhuber and Poggio, 1999) and it is unclear how sensorimotor information would be incorporated. Recent work from our lab (Hawkins et al., 2017) proposed that the neocortex processes a sensorimotor sequence by converting it into a sequence of *sensory features at object-centric locations*. The neocortex could then learn and recognize objects as sets of these sensory features at locations. This approach integrates movement into object recognition, and forms representations of objects that generalize over novel sequences of movement. However in that paper we left open the neural mechanisms for computing such a location signal.

In this paper, we show how the neocortex could compute these object-centric locations. With this missing piece filled in, we present a neural network model that learns to recognize static objects, receiving only a sensorimotor sequence as input.

Representing the locations of sensed features in object centric coordinates makes object recognition very efficient, but it requires a complex computation (Marr and Nishihara, 1978). The system must establish a coordinate system before it can begin representing the locations of sensed features. We show how part of this problem is simplified by using a conception of *location* that is inspired by grid cells. Recognizing an object requires establishing the directions of the axes of the coordinate system, but it doesn't require choosing an origin for the coordinate system. Our proposed model recognizes objects using the relative locations of features instead of using their locations relative to some origin.

Grid cell location representations fit naturally into neural mechanisms for processing sensorimotor sequences. Our model assigns each new object its own unique set of locations by first activating a random location representation and then updating it with each motor input. It recognizes objects by using each sensory input to recall and refine possible locations, and by using each motor input to update the possible locations.

We review the basic properties of grid cells and we propose that the neocortex uses analogs of grid cells to model objects just as the hippocampal formation uses them to model environments. Building on the theoretical framework introduced in (Hawkins et al., 2017) we propose that every neocortical column contains a variant of this model. Based on the cortical anatomy, we propose that cells in Layer 6 employ grid cell like mechanisms to represent object specific locations that are updated through movement. We propose that Layer 4 uses its input from Layer 6 to predict sensory input.

How Grid Cells Represent Locations and Movement

We first review how grid cells in the entorhinal cortex represent space and location. Although many details of grid cell function remain unknown, general consensus has emerged for a number of principles. Here we focus on two properties that are critical to our model: location coding and path integration.

Individual grid cells become active at multiple locations in an environment, typically in a repeating triangular lattice that

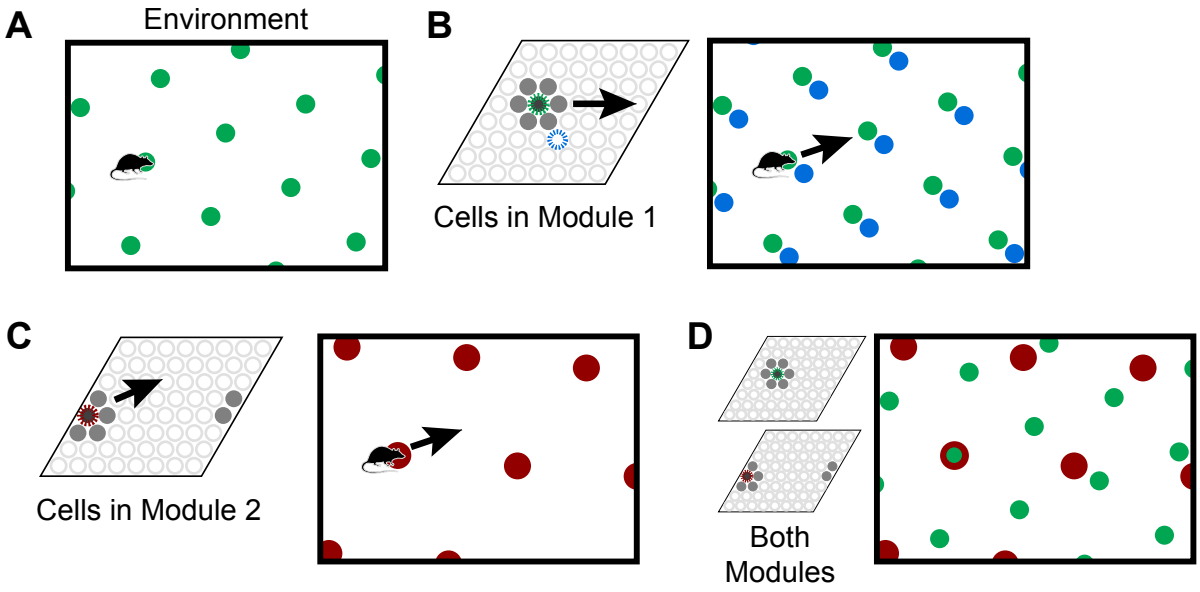


Figure 1. Grid cells represent locations in environments. **(A)** An individual grid cell becomes active at multiple locations (green circles) in an environment. The locations of activation form a repeating grid-like lattice. **(B)** A grid cell module (left) is a set of cells that share the same lattice scale and orientation but which activate at different relative positions in the environment. If you sort the cells by their relative firing locations, it forms a rhombus-shaped tile. As the animal moves, as shown by the arrow, a bump of cell activity will move in some direction through this rhombus. Two grid cells and their firing locations (green and blue) are highlighted. The grid cell module will activate cells at every location in an environment, but because of tiling, a single grid cell module cannot represent a location uniquely. **(C)** This figure shows how a second module tiles the same space differently. Each cell's firing fields have a larger scale and a different orientation than the module in (A) and (B). The same movement of the animal as shown by the arrow causes the bump to move in a different direction and a different distance than the bump in the first module in (B). In this case, the bump overlaps the edge of the rhombus, so it wraps around. **(D)** Although a single module cannot represent locations in an environment uniquely, the activity across multiple modules can. Here we superimpose the firing patterns of the two modules. Note that when the green and red cells fire together, only one location is possible. The larger the number of modules, the more locations that can be represented uniquely.

resembles a grid (**Figure 1A**). The side length of these triangles in is known as the grid cell's "scale". A grid cell "module" is a set of grid cells that activate with the same lattice scale and orientation but different positions, such that one or more grid cells will be active at any location (**Figure 1B**). If you sort the grid cells in a module by their relative firing locations, they form a rhombus-shaped tile. As the animal moves, a "bump" of activity moves across this rhombus (**Figure 1B and 1C**). The activity in a single module provides information on an animal's location, but this information is ambiguous; many locations within the environment can lead to the same activity.

To form a unique representation requires multiple grid cell modules with different scales or orientations (**Figure 1C and 1D**). For illustration purposes say we have 10 grid cell modules and each module can encode 25 possible locations via a bump of activity. These 10 bumps encode the current location of the animal. Notice, if the animal moves continuously in one direction the activity of individual modules will repeat due to the tiling, but the ensemble activity of ten modules is unlikely to repeat due to the differences in scale and orientation between the modules. The representational capacity formed by such a code is large. In our example the number of unique locations that can be represented is $25^{10} \approx 10^{14}$. A review of the capacity and noise robustness of grid codes can be found in (Fiete et al., 2008; Sreenivasan and Fiete, 2011).

As an animal moves, the active grid cells in a module change to reflect the animal's updated location. This change occurs even if the animal is in the dark (Hafting et al., 2005), telling us that grid cells are updated using information about the animal's movement. This process, called "path integration", has the desirable property that regardless of the path of movement, when the animal returns to the same physical location, then the same grid cells will be active (**Figure 2A**). Path integration is imprecise so in learned environments sensory landmarks are used to "anchor" the grid cells and prevent the accumulation of path integration errors (McNaughton et al., 2006; Ocko et al., 2018).

A final important property is that the location representations can be unique to each environment. Suppose that upon first entering a new environment each grid cell module activates a random bump to represent the current location. Then all the location representations that the animal can move to in that environment will, with high probability, be unique to that environment. The initial random starting point thus implicitly defines a unique location space for each environment, including locations that have not yet been explicitly visited. Since each module independently integrates motion information, path integration properties automatically hold for each new environment. Consequently, path integration can be learned once for each module and then reused across all environments. The location space for each new environment will be a tiny subset of the full space of possible cell activities (in our

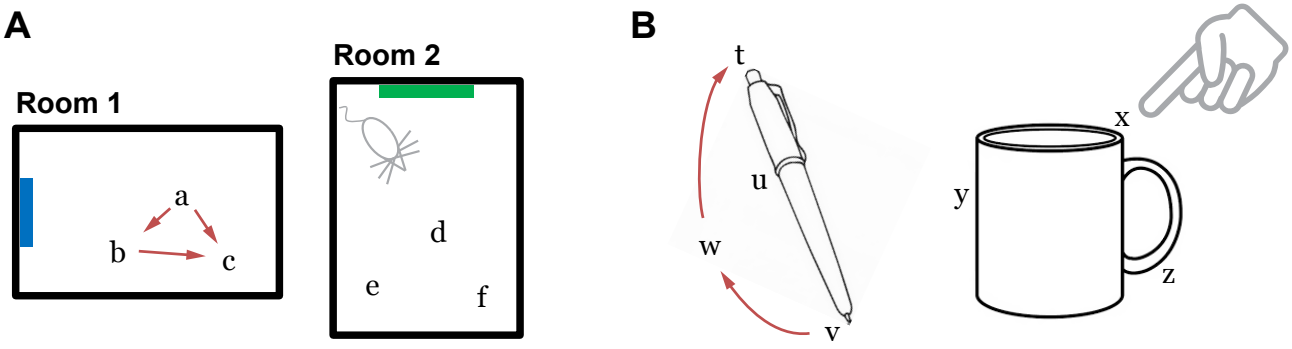


Figure 2. (A) Grid cells in the entorhinal cortex represent locations of a body in an environment. The location representations are updated by movement (Room 1). The path integration property ensures that the representation of location c is independent of the path taken to get there. Locations are unique to each environment such that the representations of locations a, b, and c are distinct from the representations of any point in Room 2. **(B)** We propose that the neocortex contains grid cell analogs that represent locations relative to an object. The location representations are unique to each object.

example above the full space contains 25^{10} points), thus the capacity for representing environments is quite large. When re-entering a previously learned environment, learned associations between sensory cues and grid cells are used to "re-anchor" or re-activate the previous location space.

To summarize the above properties, a set of grid cell modules can unambiguously represent locations in an environment. These locations can be path integrated via movement. By choosing random starting points within modules, unique location spaces can be defined for each environment. The space of all possible cell activations grows exponentially with the number of modules, thus the capacity for representing locations and environments is large.

MODEL

We propose that grid cell equivalents exist throughout the neocortex. Rather than representing the location of the animal in an environment, we propose that cortical grid cells represent the location of sensory patches, for example the tip of a finger, in the reference frame of an object (**Figure 2B**). Similar to traditional grid cells, cortical grid cells define a unique location space around each object. As a sensor moves, populations of grid cells representing each sensory patch's location will path integrate through unique location spaces. The relative locations of features on an object can thus be used as a powerful cue for disambiguating and recognizing objects.

Our network model integrates information over sensorimotor sequences, associating unique location spaces with objects and then identifying these location spaces. To outline the mechanism, let us first consider the question: how might a rat recognize a familiar environment? It must use its sensory input, but a single sensory observation is often insufficient to uniquely identify the environment. The rat thus needs to move and make multiple observations.

To combine information from multiple sensory observations, the rat could use each observation to recall the set of all locations associated with that feature. As it moves, it would then perform path integration to update each possible location. Subsequent sensory observations would be used to narrow down the set of locations and eventually disambiguate the location.

Our model uses this strategy to recognize objects with a moving sensor. According to this model, when you sense an object, the sensed feature causes you to recall locations where you've sensed this feature before. This is represented by a superposition of these previously learned locations. As you move your sensor, the network performs path integration on each of these recalled locations, i.e. the movement signal shifts the activity within each grid cell module. With subsequent sensations, the network will narrow down this list of locations until it uniquely identifies a specific location on a specific object that is consistent with the sequence of sensations and movements.

How does the animal use its sensory input to recall locations? It's unclear exactly how animals learn environments but sensory features are known to invoke grid cell activity associated with familiar environments (Barry et al., 2007). During learning our model associates sensory input with the currently active grid cells at each location.

How does the animal represent and perform path integration on an ambiguous location? In our model, the grid cell modules are capable of having multiple simultaneous bumps of cell activity. We refer to this set of simultaneously-active representations as a *union* of locations. The system is capable of path integrating unions of locations; during movement, every active bump in a module is shifted.

Model Description

Our two-layer model consists of two populations of neurons and four primary sets of connections **(Figure 3)**. For each movement of the sensor, the network goes through a progression of stages, processing the motor input followed by the sensory input. Each stage corresponds to using the connections from one of the numbered arrows in **Figure 3**. We show an example of the network going through these stages three times in **Figure 4**.

Stage 1. Motor input arrives before the sensory input and is processed by the location layer, which consists of grid cell modules. If this layer has an active location representation, it uses the motor

input to shift the activity in each module, computing the sensor's new location.

Stage 2. This updated grid cell activity propagates to the sensory layer and causes a set of predictions in that layer.

Stage 3. The sensory layer receives the actual sensory input. The predictions are combined with sensory input. The new activity is a union of highly sparse codes. Each sparse code represents a single sensory feature at a specific location that is consistent with the input so far.

Stage 4. The sensory layer activity propagates to the location layer. Each module activates a union of grid cells based on the sensory representation. The location layer will contain a union of representations of locations that are consistent with the input so far.

After the fourth stage the next motor action is initiated and the cycle repeats. The next few sections describe the network structure and each of these 4 stages in detail, as well as the learning process.

Notation and network structure

We compute the network activity through a set of discrete timesteps. Each time step t consists of a progression of the 4 stages. Each neuron in the network has a binary output; a cell is either active or inactive.

The location layer consists of a set of independent grid cell modules. The active cells of module i at time t are denoted by the binary array $\boldsymbol{A}_t^{\mathrm{loc},i}$. The layer activity $\boldsymbol{A}_t^{\mathrm{loc}}$ consists of the concatenation of all of the module activities $\boldsymbol{A}_t^{\mathrm{loc},i}$. During inference, activity in the location layer is updated twice per timestep, once in response to movement and once in response to sensory-derived input. Where necessary, we denote these two vectors as $\boldsymbol{A}_{t,\mathrm{move}}^{\mathrm{loc}}$, and $\boldsymbol{A}_{t,\mathrm{sense}}^{\mathrm{loc}}$. Each module's bump of activity is randomly initialized for each new object, and the concatenated activity thus represents object-specific locations.

The sensory layer represents the sensed feature, and this representation is specific to an object-centric location. The layer is organized into a set of mini-columns such that all the cells in a mini-column share the same feedforward receptive fields and respond to the same sensory input (Hawkins and Ahmad, 2016). The active cells of mini-column i are denoted by the binary array $\boldsymbol{A}_t^{\mathrm{in},i}$. The layer activity $\boldsymbol{A}_t^{\mathrm{in}}$ consists of the concatenation of all of the mini-column activities $\boldsymbol{A}_t^{\mathrm{in},i}$.

This model takes advantage of properties of active dendrites (Major et al., 2013). Every cell in each layer has a set of distal dendrites which are split into segments that each independently learn and activate in response to coincident patterns. $\boldsymbol{D}_{c,d}^{\mathrm{loc}}$ and $\boldsymbol{D}_{c,d}^{\mathrm{in}}$ each denote a vector which specifies the synapses of dendrite d on cell c in the location layer and sensory layer, respectively. The synapse weights are either 0 or 1.

The location layer projects to the distal dendrites of the sensory layer (**Figure 3**, connection 2). Thus $\boldsymbol{D}_{c,d}^{\mathrm{in}}$ is a vector with the same length as $\boldsymbol{A}_t^{\mathrm{loc}}$ where a 1 represents a connection to a cell in the location layer. These connections have a modulatory effect (see below). The sensory layer projects to the distal dendrites of the location layer (**Figure 3**, connection 4). Thus $\boldsymbol{D}_{c,d}^{\mathrm{loc}}$ is a vector with the same length as $\boldsymbol{A}_t^{\mathrm{in}}$ where a 1 represents a connection to a cell in the sensory layer. These vectors are generally extremely sparse as they connect to sparsely active cells during learning (see section on learning below).

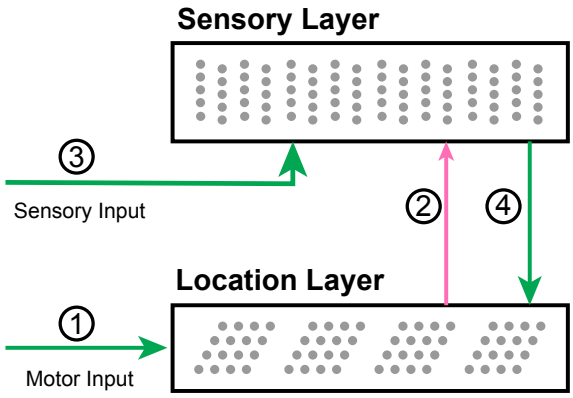


Figure 3. A diagram of the network with arrows indicating the main connections. (1) Motor input shifts the activity in the location layer. (2) The active location cells provide modulatory predictive input to the sensory layer. (3) Sensory input activates cells in the sensory layer. (4) The location is updated by the new sensory representation.

In each timestep, dendritic segments that receive sufficient input exhibit a dendritic spike. Cells with dendritic spikes are denoted with the binary vectors $\boldsymbol{\pi}_t^{\mathrm{in}}$ and $\boldsymbol{\pi}_t^{\mathrm{loc}}$. These denote whether each cell was *predicted* from the other layer's activity. Designating θ^{in} and θ^{loc} as spike thresholds, and using the existential quantifier $\exists$,

$$\boldsymbol{\pi}_t^{\mathrm{in},c} = \begin{cases} 1, & \exists_d[\boldsymbol{D}_{c,d}^{\mathrm{in}} \cdot \boldsymbol{A}_{t,\mathrm{move}}^{\mathrm{loc}} \geq \theta^{\mathrm{in}}] \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$\boldsymbol{\pi}_t^{\mathrm{loc},c} = \begin{cases} 1, & \exists_d[\boldsymbol{D}_{c,d}^{\mathrm{loc}} \cdot \boldsymbol{A}_t^{\mathrm{in}} \geq \theta^{\mathrm{loc}}] \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Stage 1: Use movement to update location layer

The location layer consists of a set of independent grid cell modules, each with its own scale and orientation. We follow notation and assumptions of other grid cell models and analyses (Ocko et al., 2018; Sreenivasan and Fiete, 2011). Within a module, the active cells are always part of a Gaussian bump of cell activity centered at a position, or *phase*, within the module's tile (**Figure 1B**). We designate the phase of module i's bump as $\vec{\phi}_t^i$. Each cell within a module is centered at a phase, and its activity at time t is proportional to its nearness to $\vec{\phi}_t^i$. The network responds to movement commands by updating $\vec{\phi}_t^i$ for each module.

Movement will shift each module's bump according to the module's scale and orientation. A 2D transform matrix $\boldsymbol{M}_i$ associated with each module represents how the module converts a movement vector for the sensor into a movement vector for the bump. Denoting the scale of module i as s_i and the orientation as θ_i, this transformation matrix is:

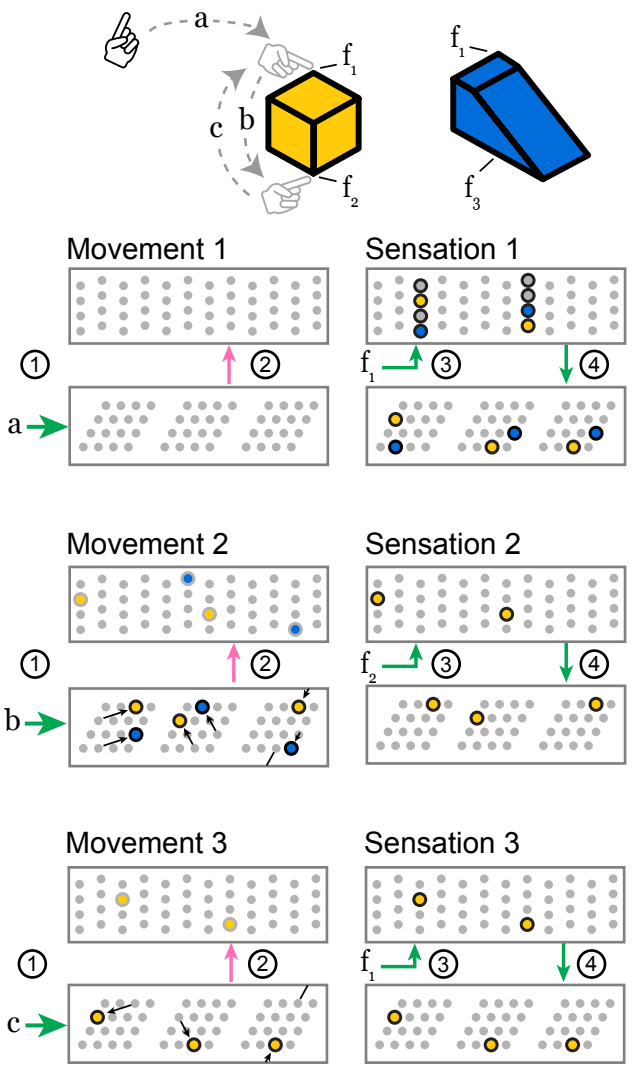


Figure 4. As the sensor moves over a previously learned object, these two layers receive a sensorimotor sequence and recognize the object. Features f_1, f_2 and f_3 indicate the sensory input invoked by touching the objects at the indicated locations. Motor commands a, b, and c indicate the motor input received by the network when the sensor makes a movement. The objects are colored to relate them to active cells below. We show three movements, each consisting of the four stages above, and we draw snapshots of the network at the end of stages 2 and 4. The stages 1 through 4 correspond to the connections in Figure 3. **Movement 1.** The network receives a movement command, and nothing happens because it doesn't have a current location representation. **Sensation 1.** The sensor senses feature f_1 which provides input to every cell in a set of mini-columns. None of the cells were predicted, so all become active. This feature has been learned on two objects, so this set of active cells contains two feature-at-location representations, shown in yellow and blue. These representations drive a pair of location representations to become active. This union of activity encodes the two possible locations that could have produced the sensation. **Movement 2.** Motor input b causes each module to perform path integration, shifting its bump according to its scale and orientation. The newly active cells provide modulatory input to the sensory layer which predicts two different potential features, f_2 and f_3, priming two representations to become active. **Sensation 2.** The sensor senses feature f_2, and only the predicted cells in the f_2 mini-columns become active. The other predicted cells do not activate. This active representation drives a single representation to become active in the location layer. At this point, the network has identified the cube. **Movement 3 and Sensation 3.** Subsequent movements maintain the unambiguous representation as long as the sensed features match those predicted by the path-integrated locations. A movement back to the original location, for instance, causes a prediction only for the f_1 representation specific to that location on the cube.

$$\boldsymbol{M}_i = \begin{bmatrix} s_i \cos\theta_i & s_i \cos(\theta_i + 60°) \\ s_i \sin\theta_i & s_i \sin(\theta_i + 60°) \end{bmatrix}^{-1} \quad (3)$$

Each module receives the same 2D movement vector $\vec{d}_t$, and it shifts its bump as follows:

$$\vec{\phi}^i_{t,\text{move}} = \left(\vec{\phi}^i_{t-1,\text{sense}} + \boldsymbol{M}_i \vec{d}_t\right) \bmod 1 \quad (4)$$

In addition to these properties, our model requires a grid cell module to be capable of path integrating multiple bumps simultaneously. Each module represents uncertainty by activating multiple bumps, one for each possible location. We refer to this as a *union* of locations. We designate a module's set of bumps as Φ^i_t. With every movement, we apply Eq. (4) to every phase in Φ^i_t.

$$\Phi^i_{t,\text{move}} = \left\{\left(\vec{\phi} + \boldsymbol{M}_i \vec{d}_t\right) \bmod 1 \;\middle|\; \vec{\phi} \in \Phi^i_{t-1,\text{sense}}\right\} \quad (5)$$

Rather than simulating individual cell dynamics explicitly, each module in the location layer simply maintains a list of activity bump phases and updates each one according to Eq. (5). The location layer then outputs the binary vector $\boldsymbol{A}^{\text{loc}}_{t,\text{move}}$ by thresholding the activity of each cell within the module (see Model Details). In the discussion we review models of individual grid cell dynamics and discuss their compatibility with unions.

Stage 2: Use updated location to form sensory predictions

In Stage 2, we compute which sensory features are predicted by the location layer. In our network these predictions are represented by $\boldsymbol{\pi}^{\text{in}}_t$, the activity of distal dendritic segments of cells in the sensory layer. We compute $\boldsymbol{\pi}^{\text{in}}_t$ from $\boldsymbol{A}^{\text{loc}}_{t,\text{move}}$ using Eq. (1) above.

Each sensory feature that has been encountered at any of the current possible locations will be predicted. Note that because $\boldsymbol{A}^{\text{loc}}_{t,\text{move}}$ is a concatenation of all grid cell modules, the predictions are based on highly specific location codes.

$\boldsymbol{\pi}^{\text{in}}_t$ has a modulatory effect on the activity of the sensory layer, as described in Stage 3.

Stage 3: Calculate activity in sensory layer
In Stage 3, the sensory layer uses the sensed feature to confirm correct predictions and to disregard incorrect predictions. When no predictions are correct, it activates all possible feature-at-location representations for the feature. This stage is responsible for both activating and narrowing unions.

The sensory layer is identical to the sensory input layer in (Hawkins et al., 2017). In this layer, all the cells in a mini-column share the same feedforward receptive fields. Each sensory feature is represented by a sparse subset of the mini-columns, denoted by $\boldsymbol{W}_t^{\text{in}}$. When a cell is predicted, it is primed to become active, and if it receives sensory input it will quickly activate and inhibit other cells in the mini-column.

The active cells within the sensory layer are selected by considering $\boldsymbol{W}_t^{\text{in}}$ and by considering which cells are predicted by the location layer, i.e. which cells have an active dendritic segment. If a cell is predicted and it is in a mini-column in $\boldsymbol{W}_t^{\text{in}}$, it becomes active. If a mini-column in $\boldsymbol{W}_t^{\text{in}}$ has no predicted cells, every cell in that mini-column becomes active.

$$\boldsymbol{A}_t^{in,i,c} = \begin{cases} 1, & i \in \boldsymbol{W}_t^{\text{in}} \text{ and } \boldsymbol{\pi}_t^{\text{in},i,c} > 0 \\ 1, & i \in \boldsymbol{W}_t^{\text{in}} \text{ and } \sum_{c'} \boldsymbol{\pi}_t^{\text{in},i,c'} = 0 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

If the location layer has uniquely identified the current location, there will be exactly one active cell in each mini-column in $\boldsymbol{W}_t^{\text{in}}$ encoding this sensory feature at the current location. If the location layer contains a union of locations, the sensory layer will represent this feature at a union of locations. Note that the location layer may predict features that are not represented in $\boldsymbol{W}_t^{\text{in}}$. These minicolumns will not contain any activity after this step (**Figure 4**, Sensation 2) and the set of possible objects is thus narrowed down.

Stage 4: Update location layer based on sensory cues
In Stage 4, the activity in the sensory layer recalls locations in the location layer. $\boldsymbol{A}_t^{\text{in}}$ is used to compute $\boldsymbol{\pi}_t^{\text{loc}}$ (Eq. (2)) which drives activity in the location layer. The list of activity bump phases in each module is replaced by a new set of bumps driven by sensory input. For each cell in the location layer that has a corresponding dendritic spike, the module activates a bump centered on that cell.

$$\Phi_{t,sense}^{loc,i} = \begin{cases} \{\vec{\phi}_c | c : \boldsymbol{\pi}_t^{\text{loc},i,c} > 0\}, \exists_c [\boldsymbol{\pi}_t^{\text{loc},i,c} > 0] \\ \Phi_{t,\text{move}}^{\text{loc},i}, \quad \text{otherwise} \end{cases} \quad (7)$$

This new set of bumps is often very similar to the previous set, as in **Figure 4,** Sensation 3. Note that this step happens during inference only. During learning, the location layer doesn't update in response to sensory input; we simply assign $\Phi_{t,\text{sense}}^{\text{loc},i} = \Phi_{t,\text{move}}^{\text{loc},i}$.

The active cells in the location layer $\boldsymbol{A}_{t,\text{sense}}^{\text{loc}}$ are computed from $\Phi_{t,\text{sense}}^{\text{loc}}$ (see Model Details).

Learning
In our model the learning process involves associating locations with sensory features, and vice versa. Learning occurs only on the distal dendritic segments. At the start of training on a new object, each module in the location layer activates a bump at a random phase. This instantiates a random location space. For the rest of training, we provide a sequence of motor and sensory inputs, calculating $\Phi_{t,\text{move}}^{\text{loc}}$ as before, shifting each module's bump with each movement.

Each sensory input is represented by a set of mini-columns $\boldsymbol{W}_{t,\text{sense}}^{\text{in}}$. Following Eq. (7) above, if this part of the object hasn't been learned yet there will be no predictions in the sensory layer and every cell in these mini-columns will become active. In this case a random cell in each active mini-column is selected as the cell to learn on, i.e. to represent this sensory input at this location. If this part of the object has been learned, there will be predictions in the sensory layer. In this case the cells corresponding to the existing active segments are selected to learn on. $\boldsymbol{A}_{t,\text{learn}}^{\text{in}}$ represents these learning cells for the current time step.

Each active cell in $\boldsymbol{A}_{t,\text{sense}}^{\text{loc}}$ and $\boldsymbol{A}_{t,\text{learn}}^{\text{in}}$ selects one of its dendritic segments d' and forms connections between this segment and each active cell in the other layer. Designating "|" as bitwise OR,

$$\boldsymbol{D}_{c,d'}^{\text{loc}} := \boldsymbol{D}_{c,d'}^{\text{loc}} \mid \boldsymbol{A}_{t,\text{learn}}^{\text{in}} \quad (8)$$
$$\boldsymbol{D}_{c,d'}^{\text{in}} := \boldsymbol{D}_{c,d'}^{\text{in}} \mid \boldsymbol{A}_{t,\text{sense}}^{\text{loc}} \quad (9)$$

SIMULATION RESULTS

We ran simulations to illustrate the ability of our model to recognize objects. We generated objects with varying numbers of shared features to test the ability of the network to disambiguate. Each object consisted of ten points chosen randomly from a four-by-four grid. We placed a feature at each point, choosing each feature randomly with replacement from a fixed feature pool. Features were shared across objects and a given feature could occur at multiple points on the same object.

For every simulation, we trained the network by visiting each point on each object once. For each point, we stored the activity in the location layer in a separate classifier. We then tested the network on each object by traversing each of the object points in random order. As the sensor traversed the object, we tested whether the location representation exactly matched the classifier's stored representation for that point on that object. If it matched this representation and if this representation was unique to this object, we considered the object to be recognized. If the network never converged to a single location representation after four complete passes over the object, or if it ever converged on a wrong location, we considered this a recognition failure. In most of our experiments, these were the two possible outcomes, but it's also possible for the network to converge on the correct location representation even if that location isn't unique to the object, for example if there aren't enough modules to create a unique code. In the sections ahead, we specifically note when this occurred.

We set the sensory layer to have 150 mini-columns and 16 cells per mini-column. Each sensory feature activated a predetermined, randomly selected set of 10 mini-columns. We varied the number of cells per module and the number of modules in the location layer. For these simulations we varied the module orientations but not the scales. The orientations were evenly spaced along the 60° range of

possible orientations. Each module used the same scale, and this scale was fixed to be less than the width of the objects. We set the dendritic thresholds θ^{loc} and θ^{in} to 8 and $\lceil n * 0.8 \rceil$, respectively, where n is the number of modules in the location layer and $\lceil\ \rceil$ is the ceiling operator.

Cell activity converges to a unique location representation.
We begin by demonstrating the cell activity in a typical recognition task, and we show how it varies as the network learns more objects. In **Figure 5** we show the actual grid cell activity for several modules in the location layer as the network sensed different objects. The network was first trained on 50 objects, each with ten features drawn from a pool of 40 features. The activity changes with each new sensation, first via path integration shifts based on the movement, followed by narrowing of the activity to only the locations consistent with the newly sensed feature. The network can take a different number of sensations to narrow to a single representation per module. Once the network has narrowed, the activity in a single module may be ambiguous but the set of active cells across all modules (only three of ten modules are shown) uniquely encode the object and location.

The recognition process always follows this template. The initial sensory input typically causes dense activation, assuming the sensory feature is not unique to a single location. With subsequent movement and sensation this activity becomes sparser and eventually converges on a single representation. In **Figure 6A** we aggregate the cell activity from **Figure 5** across all objects and all modules to show the average cell activation density after each sensation. As the network learns more objects, the initial density and the convergence time increase because the network recalls more locations-on-objects for each sensory input, and it has to disambiguate between more objects.

The model approximates an ideal observer.
Ideally, the network should recognize an object as soon as it receives a sensorimotor sequence that uniquely matches that object. In **Figure 6B** we compare the recognition time with such an ideal observer. The ideal detector stores both the features and their relative locations during training and exhaustively checks the current sensorimotor sequence against the stored objects during testing. It yields a correct classification as soon as the object is unambiguous based on the features and relative locations sensed up to that point. We also include a bag of features model. It ignores location information and yields a correct classification if it can unambiguously determine the object based solely on the sensory features.

This network uses 10 modules, and the dataset contains 100 objects with 10 unique features. Very few objects can possibly be determined by a single sensation, but three or four sensations are sufficient. As we increase the number of cells in each module, the model's performance approaches that of the ideal detector. With 40x40 cells per module, the two are near identical. The bag of features model often cannot uniquely identify the objects because many objects are different arrangements of identical sets of features.

Near the model's capacity limits, recognition time degrades.
As this model learns more objects, it eventually begins taking longer to recognize objects than the ideal observer. If it's pushed further, it eventually begins failing to recognize objects. In **Figure**

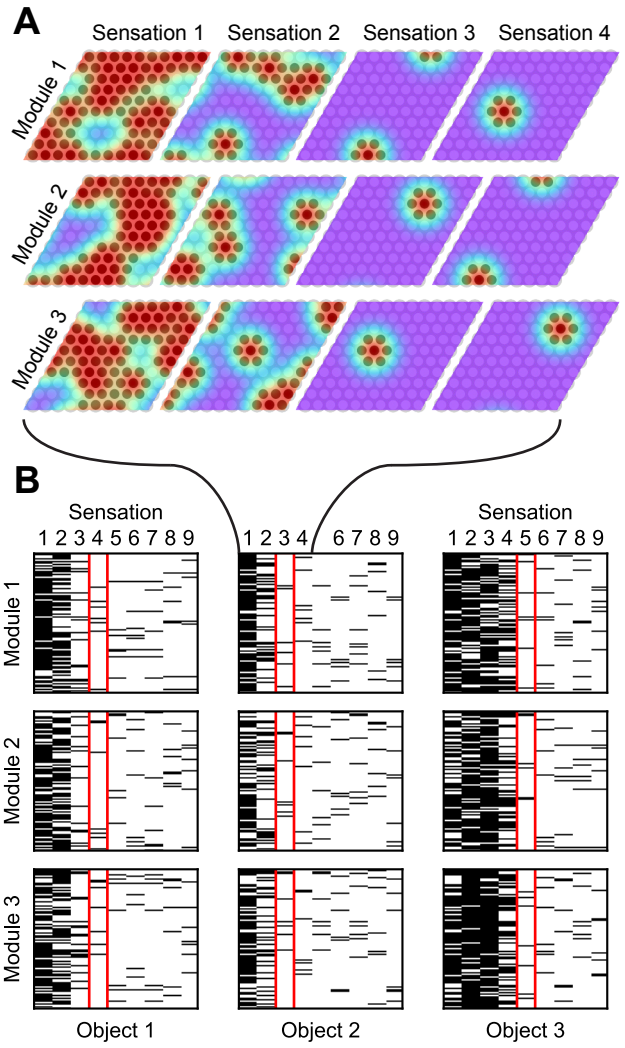


Figure 5. As the network recognizes an object, it converges onto a sparse activation. **(A)** Location layer cell activity in three (out of ten) grid cell modules while recognizing a single object. The bumps of activity are shown in color; red indicates a high firing rate. The location representation, shown as darkened circles, consists of cells that are sufficiently active. Each movement shifts the activity and each sensation narrows the activity to the cells that predict the new sensation. Cell activity converges onto a sparse representation by Sensation 3 and remains sparse afterward. **(B)** Cell activity in the same three modules as (A), shown for additional sensations and for two additional objects. Each module has 100 cells. The black lines indicate that the cell is active during the indicated sensation. After the first sensation, the location codes are very ambiguous in all cases. Depending on how common the sensed features are, the module activity narrows at different rates for the different objects. The sensation highlighted in red shows the first step in which the object is unambiguously determined. From this point on, the module activity shifts with each movement but remains unique to the object being sensed. (These simulations used a unique feature pool of size 40.)

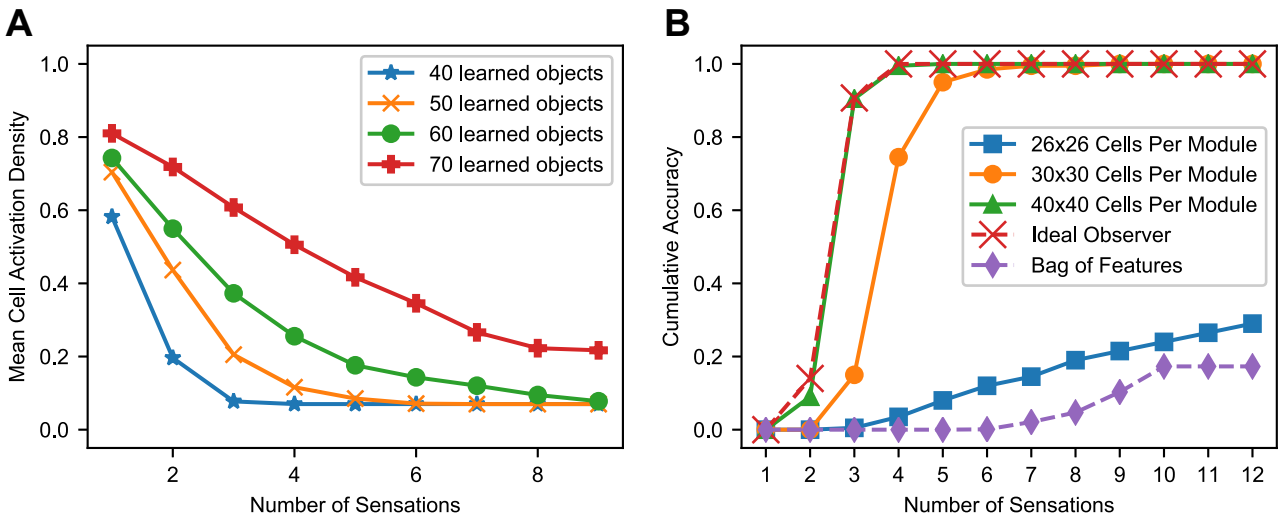


Figure 6. (A) With multiple sensations, the location layer activity converges to a sparse representation. Using the same simulation from Figure 5, we show the activation density after each sensation, averaged across all objects and modules. With additional sensations, the representation becomes sparser until the object is unambiguously recognized. With more learned objects, the network takes longer to disambiguate. The activation density of the first sensation increases with the number of learned objects. If the initial activation density is low, the network converges very quickly. If it's high, convergence can take longer. **(B)** Comparison of this network's performance with the ideal observer and a bag-of-features detector. Each model learned 100 objects from a unique feature pool of size 10. We show the percentage of objects that have been uniquely identified after each sensation, averaged across all objects for ten separate trials. The ideal model compares the sequence of input features and locations to all learned objects while the bag-of-features model ignores locations. With a sufficiently large number of cells per module, the proposed neural algorithm gets similar results to the ideal computational algorithm.

6B, the network with 30x30 cells per module requires more sensations to narrow down the object than the ideal observer, but it always recognizes the object eventually. The network with 26x26 cells per model often never recognizes the object after many sensations, indicating the network has been pushed beyond its capacity.

To understand the way that the system reaches a capacity limit, consider the density of activation in the location modules when a single feature is sensed. If a single feature occurs in many locations, then during learning the location layer and sensory layer reciprocally associate many cells in each module with that feature. Sensing that feature will cause a large percentage of the cells in the location layer to activate. If this percentage is too high, location representations that aren't supposed to be active will be largely contained in this dense activation, resulting in false positives. In the worst case, the location layer will fail to extract anything useful out of this sensory input, because it will activate nearly every location representation as part of its dense activation. We found that the main influence on the model's recognition time is whether the model is approaching its capacity limit. In the following section we characterize this capacity limit.

Capacity varies with size of location layer, statistics of objects. The previous simulations investigated the time to converge onto a unique location. Here we consider the capacity of the network independent of convergence time. We compute the fraction of objects correctly classified after many sensations, and we define the capacity as the maximum number of objects the network can store while maintaining a 90% accuracy rate. While we use 10 locations per object for these simulations, the total number of locations (number of objects times the number of locations per object) is what matters.

The model begins running into capacity limits when the sensory input causes the location layer to activate a union of representations that is large enough to cause false positives in the sensory layer. This occurs when the union contains large portions of location representations that aren't supposed to be active. The likelihood of this event is influenced by the model's number of modules, the number of cells per module, and the statistics of objects. In **Figure 7**, we characterize the impact of these variables on the model's capacity.

The model's capacity increases with the number of modules, though with diminishing returns. In **Figure 7A** we plot two lines to dissect the relationship between the number of modules and capacity. The top line shows the impact of having multiple modules, then the bottom line shows how this effect is reduced by our model's lower dendritic thresholds. Even with a single module, the network is able to converge onto one location representation for a considerable number of learned objects. However, this location representation isn't unique to the object, so it doesn't qualify as recognizing the object; we denote this with a red 'x'. Adding a few more modules ensures unique locations, and each additional module helps the model deal with large unions. They guard the model from having too many false positives when a large percentage of cells are active. More precisely, for each representation, the percentage of the cells that will be active due to randomness will be approximately equal to the activation density, with some variance, and having more modules reduces this variance. If the threshold is 100%, then the model can increase its capacity indefinitely by adding modules.

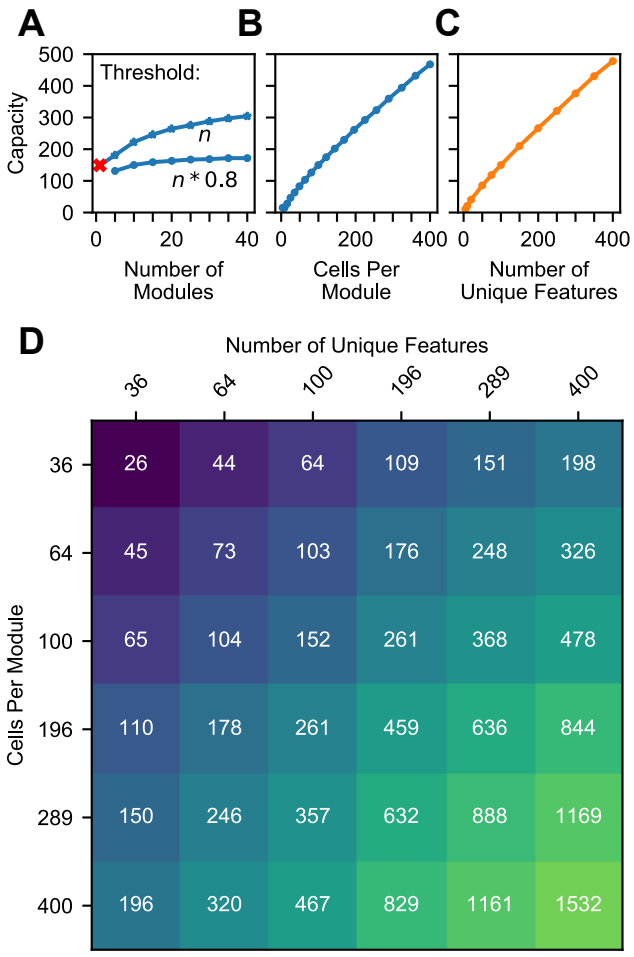


Figure 7. (A-C) Model capacity changes with model parameters (blue) and object parameters (orange). The network's capacity is the maximum number of objects that it can learn while maintaining the ability to uniquely identify at least 90% of the objects. **(A)** Increasing the number of modules in the network increases capacity sublinearly, and this impact depends on the dendritic threshold of the neurons reading this location representation. The two lines show two different thresholds relative to n, the number of modules. With only one module the network can successfully converge to one location, but that location isn't object-specific (red 'x'). **(B)** Increasing the number of cells increases capacity linearly. **(C)** Similarly, increasing the feature pool size also increases capacity linearly. **(D)** A heatmap showing the capacity of the network while varying the feature pool size and location module size**.** An increase in the number of cells per module can compensate for a decrease in the number of unique features.

However, to avoid depending on every neuron reliably firing, this model doesn't use such a high threshold. In our model, neurons will detect a location if 80% of its cells are active. With this lower threshold the benefit of additional modules asymptotes, and no number of modules will be able to handle more than 80% activation density.

Previous analysis of grid cell codes (Fiete et al., 2008) showed that the code's representational capacity increases exponentially with the number of modules. This holds true for this model, but this model's bottleneck is not the size of its unique location spaces. This model's performance depends on its ability to unambiguously represent multiple locations simultaneously. A grid cell code's *union* capacity doesn't scale exponentially with number of modules.

The model's capacity increases linearly with number of cells per module **(Figure 7B)**. As we add additional cells, the size of a bump remains constant relative to the cells, so the bump shrinks relative to the module. Because each bump activates a smaller percentage of the cells in a module, a module can activate more bumps before the network reaches the density at which object classification starts failing.

The model's capacity increases linearly with the number of unique features **(Figure 7C)**. This happens because it reduces the expected total number of occurrences of each feature, and hence the number of elements in each union. This indicates that the statistics of the world influence the capacity of the model, and it also means that this network can improve its capacity by adjusting the "features" that it extracts from sensory input.

Because these two latter parameters have independent linear relationships with capacity, they can compensate for each other. In **Figure 7D** we plot object capacities in a network with 10 modules. We show that increasing (decreasing) the number of cells per module and decreasing (increasing) the number of unique features by the same factor causes the capacity to remain approximately constant. This is illustrated by the approximate symmetry across the chart's diagonal.

The model recognizes an object if the object has at least one sufficiently uncommon feature.

Up to this point, we've characterized this model's ability to recognize objects by stating, "The network will reliably recognize an object if the network has learned fewer than c total objects," and we've measured c. This characterization is built on many assumptions about objects. It's desirable to be able to characterize the model in a way that isn't specific to these assumptions.

Given that the model's ability to recognize objects depends on the density of cell activity invoked by sensory features, we found we could characterize the model's performance more directly by answering, "The network will reliably recognize an object if the object contains a feature with fewer than k total occurrences across all learned objects," and measuring k. In another set of experiments (**Figure S1**), we generated objects using multiple alternate distributions of features. We found that the network's breaking point relative to c (the number of learned objects) did indeed vary widely with the choice of feature distribution, while its breaking point relative to k (the number of locations recalled by sensing a feature) was much more consistent across distributions. This suggests that the network's performance relative to k will hold true with real-world statistics. Using both of these metrics, c and k, we summarize all of these results in **Figure 8**.

We conclude that this model reliably recognizes an object if the object has at least one sufficiently uncommon feature, a feature that causes the network to recall a sufficiently small number of locations. After sensing this feature, the model can use other, more

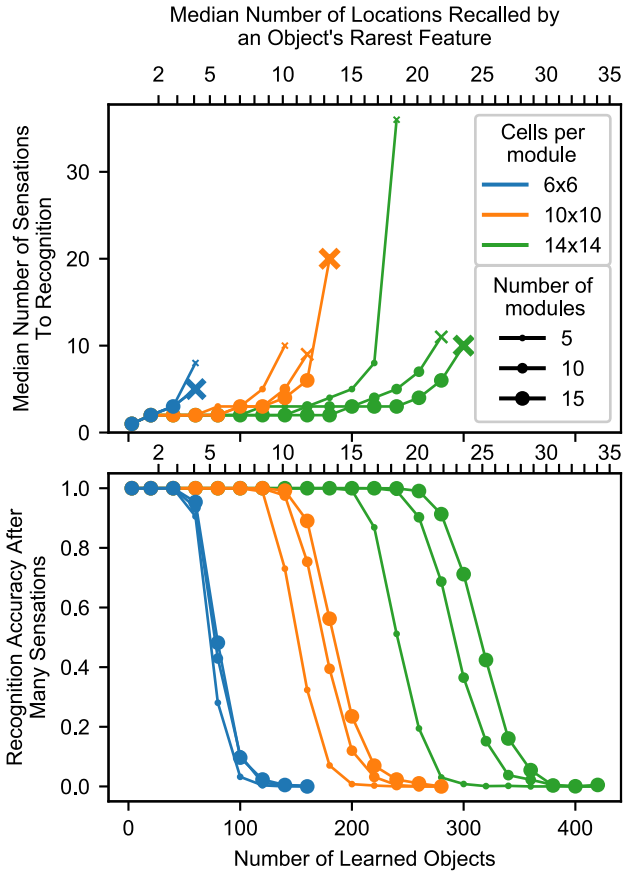


Figure 8. Summary chart showing recognition time and capacity while varying the model parameters. The top chart shows the median number of sensations required to recognize an object, and it marks the final point where a network recognized at least 50% of objects using an 'x'. The bottom chart shows the percentage of objects that were recognized after traversing every point on the object multiple times. The charts share an x axis, and this x axis has two sets of ticks and labels. The bottom ticks are numbers of learned objects, given 10 locations per object and 100 unique features. The top ticks are the typical smallest union sizes that will be invoked by each object. With different statistics, these curves will shift relative to the bottom ticks, but they should stay approximately fixed relative to the top ticks.

common features to finish recognizing the object, but it needs some initial clue to help it activate a manageable union that it can then narrow. With enough cells, this initial clue doesn't need to be especially unique. For example, the feature could have 10 learned locations-on-objects if the module has 10x10 cells per module and 10 modules, as shown in **Figure 8**, and this breaking point can be pushed arbitrarily high by adding more cells. The network can also avoid running into this breaking point by tweaking the set of "features" that it extracts from sensory input.

MAPPING TO BIOLOGY

We have described a two-layer network model for sensorimotor inference and now consider how this network motif maps to known cortical anatomy and physiology. Layer 4 (L4) is well understood to be the primary target of thalamocortical sensory inputs (Douglas and Martin, 2004; Jones, 1998; Theyel et al., 2010). These connections are believed to be driving inputs (Viaene et al., 2011) and target both excitatory and inhibitory neurons, with a slight delay for the inhibitory neurons that provides a window of opportunity for neurons to fire (Harris and Shepherd, 2015). Mountcastle identified the organization of neurons into minicolumns with shared receptive fields (Buxhoeveden, 2002; Mountcastle, 1957).

The connections between L4 and layer 6a (L6a) closely resemble the connections in our model (**Figure 3**, arrows 2 and 4). Thalamic input forms a relatively small percentage of L4 synapses; approximately 45% of its synapses come from L6a (Binzegger et al., 2004). The connections from L6a are weak (Harris and Shepherd, 2015; Kim et al., 2014). They connect to distal dendritic segments of L4 cells whereas thalamocortical afferents connect more proximally (Ahmed et al., 1994; Binzegger et al., 2004). L4 cells also form a significant number of connections to cells in L6a (Binzegger et al., 2004). These biological details closely match our network model, where the location layer (putatively L6a) has a modulatory influence on the sensory input layer (putatively L4) which in turn can drive representations in the location layer.

Our location layer also requires a motor input. Experiments show that L5 cells in motor regions, such as M2, project to sensory regions, including layer 6 (Leinweber et al., 2017; Nelson et al., 2013). There is also a potential indirect pathway through thalamocortical inputs that target layer 6 (Harris and Shepherd, 2015; Thomson, 2010). Thalamic relay cells receive input from layer 5 neurons that are presumed to be efference copies of motor commands sent subcortically (Chevalier and Deniau, 1990; Jones, 1998). Any of these direct or indirect pathways could serve as motor signals for path integration in L6.

Our model draws inspiration from the grid and place cell systems in the hippocampal formation. Our location layer is modeled after grid cells. These cells project (Zhang et al., 2013) to place cells (O'Keefe and Dostrovsky, 1971) in the hippocampus. Areas of hippocampus containing place cells also project back to areas of entorhinal cortex containing grid cells (Rowland et al., 2013). Many place cells seem to represent item-place pairs, and these pairs are learned through experience (Komorowski et al., 2009), a phenomenon that is analogous to the learning of feature-location pairs in our sensory layer.

Location representations in neocortex similar to grid cells are speculative but there is initial experimental support for grid-like codes in neocortex. fMRI experiments with humans performing tasks have led to activity signatures in prefrontal cortex that are similar to grid cell signals (Constantinescu et al., 2016; Julian et al., 2018). Direct cell recordings have also shown grid-like activity in frontal cortex (Jacobs et al., 2013). These experiments are consistent with the hypothesis that grid-like cells are present in the neocortex.

In our model, primary sensory cortex represents the sensory input at locations in an external reference frame. This is consistent with results from (Saleem et al., 2018). As a mouse ran on a virtual track, the majority of recorded cells in primary visual cortex encoded the animal's location on the track, even when the mouse received visual input that occurred at multiple points of the track. According to our

model, in this task the visual cortex represented the location of a sensor (the mouse's eye) relative to an object (the virtual track), and the cortex used the visual input to recall locations on the track while using the mouse's movement to update these location representations.

Although additional experimental work is required, evidence suggests that L4 and L6a provide the best candidate populations for our sensory and location layers, respectively.

DISCUSSION

We have presented a two layer neural network model for sensorimotor object recognition. By processing sensorimotor sequences, it learns objects as spatial arrangements of sensory features. It can recognize learned objects from novel sensorimotor sequences.

The location layer contains modules of cortical grid cells. These cells operate similarly to grid cells in the medial entorhinal cortex. They have motor inputs that shift the activity in each module based on the module's scale and orientation. And they have sensory-derived inputs that activate locations where the input has been previously learned. Path integration updates the current location on the object while sensory-derived inputs narrow down possible locations and correct errors from path integration.

The sensory layer combines location codes with sensory input to create representations of sensory inputs that are unique to objects and locations.

The model provides a concrete implementation of the location signal in our earlier model (Hawkins et al., 2017) but there are a number of practical and theoretical aspects of the model that require further research. There are remaining biological questions about the neural basis for grid cells, extensions for handling orientation, and more. The remainder of this section discusses these topics.

The cell dynamics of grid cell modules

In this model, we treated a grid cell module as a black box with well-known outside properties and unspecified internal neural circuitry, and we simulated the outside properties. We gave this population an additional property that is not typically noted in grid cells: support for unions. The grid cell modules in our model can activate and shift multiple bumps of activity. Various models of grid cell dynamics could be plugged into this model, but this union property introduces a new requirement for these models.

A few different classes of model of grid cell dynamics have been proposed (Giocomo et al., 2011). Recurrent grid cell models have received more empirical support (Yoon et al., 2013) than models in which grid cells establish their responses independent of each other. In recurrent models, grid cells determine their activity using velocity input and connections to other grid cells, either directly or via interneurons. A well-known recurrent model is the continuous attractor network (Burak and Fiete, 2009) which performs robust path integration and offers a simple structural explanation for the origin of the hexagonal firing fields. Another explanation for the origin of these fields is that they are an optimal code for locations which is naturally learned by neural learning rules. This argument has appeared in two lines of research. In path integration models, (Banino et al., 2018) and (Cueva and Wei, 2018) found that recurrent neural networks trained to perform path integration naturally develop grid cells, although neither report the network developing the full rhombus of grid cells at each scale. Setting path integration aside, (Kropff and Treves, 2008), (Dordek et al., 2016), and (Stachenfeld et al., 2017) showed that cells performing Hebbian learning on place cell activity would naturally learn periodic firing fields similar to those of grid cells.

The continuous attractor network is so named because it has a continuous manifold of stable states. If the network activates a representation that isn't within this manifold of stable representations, the activity will move to the nearest stable state. In typical continuous attractor networks, a union is not a stable state, and the network will collapse a union of bumps into a single bump. It's an open question whether it's possible to have a continuous attractor with stable union states. Unions may not be compatible with attractor dynamics. In this work we've shown that it's theoretically appealing for the grid cell module to be able to shift activity around the module without these constrictive attractor dynamics. The attractor dynamics are appealing in part because they explain the hexagonal firing fields, but as mentioned, those may be explainable as the natural result of a recurrent neural network learning a location code.

Modeling this network at a lower level of abstraction is an area for future research.

Representing the orientation of objects

With this model, we showed how cortex could use principles of grid cells to compute the location of a sensor in the reference frame of the sensed object. Notably, this method solved the problem without needing to go through a process of choosing an origin on the object. However, in this formulation, it's still necessary to infer the directions of the axes of the object's reference frame, and this model doesn't yet address this point. Because this model ignores orientations, it will only recognize an object if the object is at its learned orientation relative to the sensor. The model would need to learn objects at every orientation, assigning each a different space of locations.

Just as this model's location representation is inspired by grid cells, an extended version of this model could represent orientation using analogs to head-direction cells (Taube et al., 1990). Grid cells represent the animal's location on a cognitive map, whereas head-direction cells seem to represent the animal's orientation relative to the cognitive map's compass rose. When an animal moves, each entorhinal grid cell module moves its bump of activity according to its direction of movement on the compass rose. Similarly, we expect the neocortex to represent the orientation of sensors in the reference frame of the sensed object, and this orientation will influence how sensor movement translates into the movement of bumps in cortical grid cell modules. Additionally, the sensor's orientation is needed to predict the sensory input. In this extended model, instead of pure location cells projecting to the sensory layer, we expect these cells to represent conjunctive locations and orientations.

Other extensions

This model uses 2D grid cell modules to model 2D objects. The entorhinal cortex's representation of 3D space is an active area of research (Jeffery et al., 2015). This model's general strategy will

work with any 3D location code that provides unique location representations at every 3D location around an object.

This model processes sensory input from a single sensory patch. The body has many sensory patches, and we predict that each of these patches has an instantiation of this network processing its input. In previous work (Hawkins et al., 2017) we showed how these networks work together by using an additional population of cells which represents the current object, invariant to the sensor's location. Each of these networks votes on this classified object. In this way, multiple instantiations of this network work together to recognize objects using the input from multiple independent moving sensors.

As shown in **Figure 7A**, with only one grid cell module this model can still infer an object-centric location, but this location isn't specific to the object, so inferring this location doesn't qualify as object recognition. However, this single-module location would still be useful in an object recognition system. In (Hawkins et al., 2017) we showed that if locations aren't object-specific, the model can still recognize objects if it includes an additional population of cells to represent the current object. This third population receives input from the sensory layer, learning an object as a set of feature-at-location representations. Additionally, if this third population projects back to the sensory layer, it helps the model infer the location.

Relationship to other models

Our model identifies objects using the relative location of sensory features. Objects are disambiguated over time through successive sensations and movements. In contrast, most existing models of object recognition involve a strictly feedforward spatial hierarchical system (DiCarlo et al., 2012; Riesenhuber and Poggio, 1999; Serre et al., 2007; Yau et al., 2009). In these models each level detects the presence of increasingly abstract features in parallel until a complete object is recognized at the top of the hierarchy. Our model implies that each level of a hierarchy might be more powerful than previously assumed. In (Hawkins et al., 2017) we discussed how spatially separated sensory inputs (across multiple cortical columns each computing a location signal) can cooperate in parallel to recognize objects, and some of the implications on hierarchy. Our model suggests a path for integrating sensorimotor behavior into a hierarchical system, and accounts for the many inter and intracortical connections that are not explained by a purely feedforward model. A more detailed study integrating our model into a full hierarchical system is a topic for future research.

There is also a rich history of sensorimotor integration and learning internal models in the context of skilled motor behavior (Wolpert et al., 2011; Wolpert and Ghahramani, 2000). These have primarily focused on learning motor dynamics and kinematic control, such as reaching and grasping tasks. Our model focuses on the more structured object recognition paradigm, but there are many high-level similarities with this body of literature. Our location layer is highly analogous to the forward models posited to exist in motor control (Wolpert et al., 2011). In both cases the current state is updated using a motor efference copy to compute the next state. In both these models, this is an estimate of the state that informs predictions (arrow 2 in **Figure 3**) and is then combined with sensory input to produce the current state (arrows 3 and 4 in **Figure 3**). The primary difference is that our model recognizes a set of structured objects rather than motion trajectories. The neural mechanisms are also significantly different. Nevertheless it is intriguing that the same ideas can be applied to both situations and may reflect a more general design pattern in the brain. An in-depth exploration of this relationship is a topic for future research.

Testable Predictions

Our model makes a number of experimentally testable predictions. We expand on the predictions from (Hawkins et al., 2017).

1. The neocortex uses analogs of grid cell modules to represent locations relative to objects. The cell activity in a module moves through a manifold of representations as the attended location moves relative to an object. For example, in somatosensory areas, cells will respond selectively when the animal's finger is at particular locations relative to an attended object. Just as entorhinal grid cell modules use the same map for every environment, the cells of a single module use the same manifold of representations for every object. This map has limited size, and hence it will perform some form of wrapping at its edges.
2. The neocortex uses a population code of multiple modules to represent object-specific locations.
3. These modules are in Layer 6 of the neocortex.
4. The projection from Layer 6 to Layer 4 modulates which cells in Layer 4 become active. If Layer 6 input is experimentally inhibited, activity in Layer 4 will become denser.
5. The connection from Layer 4 to Layer 6 can drive the Layer 6 cells to become active, but this only occurs when the animal receives an unpredicted input.

MODEL DETAILS

Each module has a fixed number of cells which each have a fixed phase in the rhombus. Gaussian bumps of activity move over these cells. A cell is considered active if its firing rate is sufficiently high. In this section, we walk through the details of these calculations.

Each module contains $w * w$ cells. Each cell c has a constant phase $\vec{\phi}_c$. We partition the 2D range $[0,1) \times [0,1)$ into $w * w$ ranges of equal area and set each cell's $\vec{\phi}_c$ to the center of one of these ranges. Because these modules use basis vectors separated by 60° (Eq. (3)), when mapped onto physical space these cells form a rhombus and they pack together in a hexagonal formation.

The normalized firing rate of a cell c caused by bump b, denoted $r_{c,b}$, is equal to the Gaussian of the distance between them.

$$\text{Gaussian}(d) = e^{-\frac{d^2}{2\sigma^2}} \tag{10}$$

$$r_{c,b} = \text{Gaussian}\left(\text{Distance}(\vec{\phi}_c, \vec{\phi}_b)\right) \tag{11}$$

$\text{Distance}(\vec{\phi}_c, \vec{\phi}_b)$ represents the shortest distance between the cell and the bump on the phase rhombus. Computing this distance requires changing the basis so that each $\vec{\phi}$ is a point on a rhombus rather than a point on the $[0,1) \times [0,1)$ square. σ specifies the size of the bump relative to the rhombus, which we discuss later.

When there are multiple bumps, the firing rates from each bump are combined as if each rate encodes a probability of an event. The combined firing rate encodes a probability of the "or" of those events.

$$r_c = 1 - \prod_b \left(1 - r_{c,b}\right) \tag{12}$$

To compute module i's active cells $\boldsymbol{A}_t^{\text{loc},i}$, we compute each cell's firing rate and check whether it is above the active firing rate r_{active}.

$$\boldsymbol{A}_t^{loc,i,c} = \begin{cases} 1, & r_c \geq r_{\text{active}} \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

We choose r_{active} using a readout resolution parameter $\delta\phi$ which is common in grid cell models (Fiete et al., 2008; Sreenivasan and Fiete, 2011).

$$r_{\text{active}} = \text{Gaussian}\left(\frac{\delta\phi}{2} * \frac{2}{\sqrt{3}}\right) \tag{14}$$

The readout resolution $\delta\phi$ approximately specifies the diameter of the range of phases that a bump encodes. Because modules are 2D, if the readout resolution is 1/4 then the bump can encode approximately 16 possible positions in the rhombus. The multiplicative factor of $2/\sqrt{3}$ accounts for the fact that when circles pack together in hexagonal formation, they leave some area uncovered; this factor expands the circles to overlap and cover this area. With a single bump, changing the σ parameter has no effect on the model, because the $\delta\phi$ parameter has complete control over what fraction of the cells are considered active. σ becomes relevant when there are multiple bumps. The wider these bumps are, the more they'll combine and cause interstitial cells' firing rates to rise above r_{active}.

We vary σ and $\delta\phi$ as follows. Our baseline parameters mimic the sparsity of rat entorhinal grid cell modules. We set σ to 0.18172, a number we obtained by fitting a 2D Gaussian to the firing fields generated by the model in (Monaco and Abbott, 2011). We set $\delta\phi$ to a conservative estimate of 1/3. In this configuration, we assign the network 6x6 cells, and a bump always activates at least 2x2 cells. When we test the network with more cells, we assume that the bump remains a fixed size relative to the cells, i.e. that the bump is smaller relative to the size of the module, and we scale down σ and $\delta\phi$ accordingly. For example, with 12x12 cells, we use $\sigma/2$ and $\delta\phi\ /\ 2$. By varying the parameters in this way, a single bump always activates between 4 and 7 cells, depending on where the bump is centered relative its local neighborhood of cells, and as we vary the number of cells we're effectively varying the number of cells that the bump *doesn't* activate.

During learning, only the cell with the highest firing rate is associated with the sensed feature. (When a sensed feature activates a cell, it activates a bump centered on the cell via Eq. (7), activating the cells around it.) This means this model's learning resolution is twice as precise as the readout resolution. Because sensed features are associated with cells that represent a range of phases, there's always some uncertainty in the phase recalled by sensory input. If the learned resolution weren't more precise than the readout resolution, the bump of active cells would need to expand to account for this uncertainty, and the effective readout resolution would be half as precise. Using fixed-sized bumps, achieving a particular readout resolution – that is, having a bump encode a range of phases with a particular diameter – requires the learning resolution to be at least twice as precise as this readout resolution.

The ideal classifier in **Figure 6B** stores all objects as 2D arrays. During inference, it uses the first sensed feature to find all possible locations on all objects with that feature, and it stores these as candidate locations. With each subsequent movement it updates all of the candidate locations. Any updated candidates that are not valid locations on objects or contain features that don't match the new sensed feature are removed from the candidate list. Once there is only a single location left, inference is successfully completed.

The bag of features detector stores a set of features for each learned object. It does not keep track of how many times features occur, just the set of unique features present somewhere on the object. During inference, another set keeps track of which features have been sensed so far. Once there is only one object that contains all of the sensed features, inference is successfully completed. If there are multiple objects that contain all features once all locations on the object being tested have been visited, then the object cannot be uniquely classified.

Code availability

All of the source code for this model and these simulations can be found at https://github.com/numenta/htmpapers.

REFERENCES

Ahmed, B., Anderson, J. C., Douglas, R. J., Martin, K. A. C., and Nelson, J. C. (1994). Polyneuronal innervation of spiny stellate neurons in cat visual cortex. *J. Comp. Neurol.* 341, 39–49. doi:10.1002/cne.903410105.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature* 557, 429–433. doi:10.1038/s41586-018-0102-6.

Barry, C., Hayman, R., Burgess, N., and Jeffery, K. J. (2007). Experience-dependent rescaling of entorhinal grids. *Nat. Neurosci.* 10, 682–684. doi:10.1038/nn1905.

Binzegger, T., Douglas, R. J., and Martin, K. A. C. (2004). A Quantitative Map of the Circuit of Cat Primary Visual Cortex. *J. Neurosci.* 24, 8441–8453. doi:10.1523/JNEUROSCI.1400-04.2004.

Burak, Y., and Fiete, I. R. (2009). Accurate path integration in continuous attractor network models of grid cells. *PLoS Comput. Biol.* 5. doi:10.1371/journal.pcbi.1000291.

Buxhoeveden, D. P. (2002). The minicolumn hypothesis in neuroscience. *Brain* 125, 935–951. doi:10.1093/brain/awf110.

Chevalier, G., and Deniau, J. M. (1990). Disinhibition as a basic process in the expression of striatal functions. *Trends Neurosci.* 13, 277–280. doi:10.1016/0166-2236(90)90109-N.

Constantinescu, A. O., OReilly, J. X., and Behrens, T. E. J. (2016). Organizing conceptual knowledge in humans with a gridlike code. *Science (80-.).* 352, 1464–1468. doi:10.1126/science.aaf0941.

Cueva, C. J., and Wei, X.-X. (2018). Emergence of grid-like representations by training recurrent neural networks to

perform spatial localization. 1–15.

DiCarlo, J. J., Zoccolan, D., and Rust, N. C. (2012). How Does the Brain Solve Visual Object Recognition? *Neuron* 73, 415–434. doi:10.1016/J.NEURON.2012.01.010.

Dordek, Y., Soudry, D., Meir, R., and Derdikman, D. (2016). Extracting grid cell characteristics from place cell inputs using non-negative principal component analysis. *Elife* 5, 1–36. doi:10.7554/eLife.10094.

Douglas, R. J., and Martin, K. A. C. (2004). Neuronal Circuits of the Neocortex. *Annu. Rev. Neurosci.* 27, 419–451. doi:10.1146/annurev.neuro.27.070203.144152.

Fiete, I. R., Burak, Y., and Brookings, T. (2008). What Grid Cells Convey about Rat Location. *J. Neurosci.* 28, 6858–6871. doi:10.1523/JNEUROSCI.5684-07.2008.

Giocomo, L. M., Moser, M. B., and Moser, E. I. (2011). Computational models of grid cells. *Neuron* 71, 589–603. doi:10.1016/j.neuron.2011.07.023.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature* 436, 801–806. doi:10.1038/nature03721.

Harris, K. D., and Shepherd, G. M. G. (2015). The neocortical circuit: themes and variations. *Nat. Neurosci.* 18, 170–181. doi:10.1038/nn.3917.

Hawkins, J., and Ahmad, S. (2016). Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. *Front. Neural Circuits* 10, 1–13. doi:10.3389/fncir.2016.00023.

Hawkins, J., Ahmad, S., and Cui, Y. (2017). A Theory of How Columns in the Neocortex Enable Learning the Structure of the World. *Front. Neural Circuits* 11, 81. doi:10.3389/FNCIR.2017.00081.

Jacobs, J., Weidemann, C. T., Miller, J. F., Solway, A., Burke, J. F., Wei, X. X., et al. (2013). Direct recordings of grid-like neuronal activity in human spatial navigation. *Nat. Neurosci.* 16, 1188–1190. doi:10.1038/nn.3466.

Jeffery, K. J., Wilson, J. J., Casali, G., and Hayman, R. M. (2015). Neural encoding of large-scale three-dimensional space—properties and constraints. *Front. Psychol.* 6, 1–12. doi:10.3389/fpsyg.2015.00927.

Jones, E. G. (1998). Viewpoint: The core and matrix of thalamic organization. *Neuroscience* 85, 331–345. doi:10.1016/S0306-4522(97)00581-2.

Julian, J. B., Keinath, A. T., Frazzetta, G., and Epstein, R. A. (2018). Human entorhinal cortex represents visual space using a boundary-anchored grid. *Nat. Neurosci.* 21, 191–194. doi:10.1038/s41593-017-0049-1.

Kim, J., Matney, C. J., Blankenship, A., Hestrin, S., and Brown, S. P. (2014). Layer 6 corticothalamic neurons activate a cortical output layer, layer 5a. *J Neurosci* 34, 9656–9664. doi:10.1523/JNEUROSCI.1325-14.2014.

Komorowski, R. W., Manns, J. R., and Eichenbaum, H. (2009). Robust Conjunctive Item-Place Coding by Hippocampal Neurons Parallels Learning What Happens Where. *J. Neurosci.* 29, 9918–9929. doi:10.1523/JNEUROSCI.1378-09.2009.

Kropff, E., and Treves, A. (2008). The emergence of grid cells: Intelligent design or just adaptation? *Hippocampus* 18, 1256–1269. doi:10.1002/hipo.20520.

Leinweber, M., Ward, D. R., Sobczak, J. M., Attinger, A., and Keller, G. B. (2017). A Sensorimotor Circuit in Mouse Cortex for Visual Flow Predictions. *Neuron* 95, 1420–1432.e5. doi:10.1016/j.neuron.2017.08.036.

Major, G., Larkum, M. E., and Schiller, J. (2013). Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.* 36, 1–24. doi:10.1146/annurev-neuro-062111-150343.

Marr, D., and Nishihara, H. K. (1978). Representation and Recognition of the Spatial Organization of Three-Dimensional Shapes. *Proc. R. Soc. B Biol. Sci.* 200, 269–294. doi:10.1098/rspb.1978.0020.

McNaughton, B. L., Battaglia, F. P., Jensen, O., Moser, E. I., and Moser, M.-B. (2006). Path integration and the neural basis of the "cognitive map." *Nat. Rev. Neurosci.* 7, 663–678. doi:10.1038/nrn1932.

Monaco, J. D., and Abbott, L. F. (2011). Modular Realignment of Entorhinal Grid Cell Activity as a Basis for Hippocampal Remapping. *J. Neurosci.* 31, 9414–9425. doi:10.1523/JNEUROSCI.1433-11.2011.

Mountcastle, V. B. (1957). MODALITY AND TOPOGRAPHIC PROPERTIES OF SINGLE NEURONS OF CAT'S SOMATIC SENSORY CORTEX. *J. Neurophysiol.* 20, 408–434. doi:10.1152/jn.1957.20.4.408.

Nelson, A., Schneider, D. M., Takatoh, J., Sakurai, K., Wang, F., and Mooney, R. (2013). A Circuit for Motor Cortical Modulation of Auditory Cortical Activity. *J. Neurosci.* 33, 14342–14353. doi:10.1523/JNEUROSCI.2275-13.2013.

O'Keefe, J., and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Res.* 34, 171–175. doi:10.1016/0006-8993(71)90358-1.

Ocko, S., Hardcastle, K., Giocomo, L. M., and Ganguli, S. (2018). Emergent Elasticity in the Neural Code for Space. *bioRxiv*, 326793. doi:10.1101/326793.

Riesenhuber, M., and Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nat. Neurosci.* 2, 1019–25. doi:10.1038/14819.

Rowland, D. C., Weible, A. P., Wickersham, I. R., Wu, H., Mayford, M., Witter, M. P., et al. (2013). Transgenically Targeted Rabies Virus Demonstrates a Major Monosynaptic Projection from Hippocampal Area CA2 to Medial Entorhinal Layer II Neurons. *J. Neurosci.* 33, 14889–14898. doi:10.1523/JNEUROSCI.1046-13.2013.

Saleem, A. B., Diamanti, E. M., Fournier, J., Harris, K. D., and Carandini, M. (2018). Coherent encoding of subjective spatial position in visual cortex and hippocampus. *Nature*. doi:10.1038/s41586-018-0516-1.

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T. (2007). Robust Object Recognition with Cortex-Like Mechanisms. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 411–426. doi:10.1109/TPAMI.2007.56.

Sreenivasan, S., and Fiete, I. (2011). Grid cells generate an analog error-correcting code for singularly precise neural computation. *Nat. Neurosci.* 14, 1330–1337. doi:10.1038/nn.2901.

Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The hippocampus as a predictive map. *Nat. Neurosci.* 20, 1643. Available at: http://dx.doi.org/10.1038/nn.4650.

Taube, J. S., Muller, R. U., and Ranck, J. B. (1990). Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis. *J. Neurosci.* 10, 420–35. doi:10.1212/01.wnl.0000299117.48935.2e.

Theyel, B. B., Llano, D. A., and Sherman, S. M. (2010). The corticothalamocortical circuit drives higher-order cortex in

the mouse. *Nat. Neurosci.* 13, 84–88. doi:10.1038/nn.2449.The.

Thomson, A. M. (2010). Neocortical layer 6, a review. *Front. Neuroanat.* 4, 1–14. doi:10.3389/fnana.2010.00013.

Viaene, A. N., Petrof, I., and Sherman, S. M. (2011). Synaptic Properties of Thalamic Input to the Subgranular Layers of Primary Somatosensory and Auditory Cortices in the Mouse. *J. Neurosci.* 31, 12738–12747. doi:10.1523/JNEUROSCI.1565-11.2011.

Wolpert, D. M., Diedrichsen, J., and Flanagan, J. R. (2011). Principles of sensorimotor learning. *Nat. Rev. Neurosci.* 12, 739–51. doi:10.1038/nrn3112.

Wolpert, D. M., and Ghahramani, Z. (2000). Computational principles of movement neuroscience. *Nat. Neurosci.* 3, 1212–1217. doi:10.1038/81497.

Yau, J. M., Pasupathy, A., Fitzgerald, P. J., Hsiao, S. S., and Connor, C. E. (2009). Analogous intermediate shape coding in vision and touch. *Proc. Natl. Acad. Sci. U. S. A.* 106, 16457–16462. doi:10.1073/pnas.0904186106.

Yoon, K., Buice, M. A., Barry, C., Hayman, R., Burgess, N., and Fiete, I. R. (2013). Specific evidence of low-dimensional continuous attractor dynamics in grid cells. *Nat. Neurosci.* 16, 1077–1084. doi:10.1038/nn.3450.

Zhang, S. J., Ye, J., Miao, C., Tsao, A., Cerniauskas, I., Ledergerber, D., et al. (2013). Optogenetic dissection of entorhinal-hippocampal functional connectivity. *Science (80-.).* 340, 44. doi:10.1126/science.1232627.

Supplementary figures

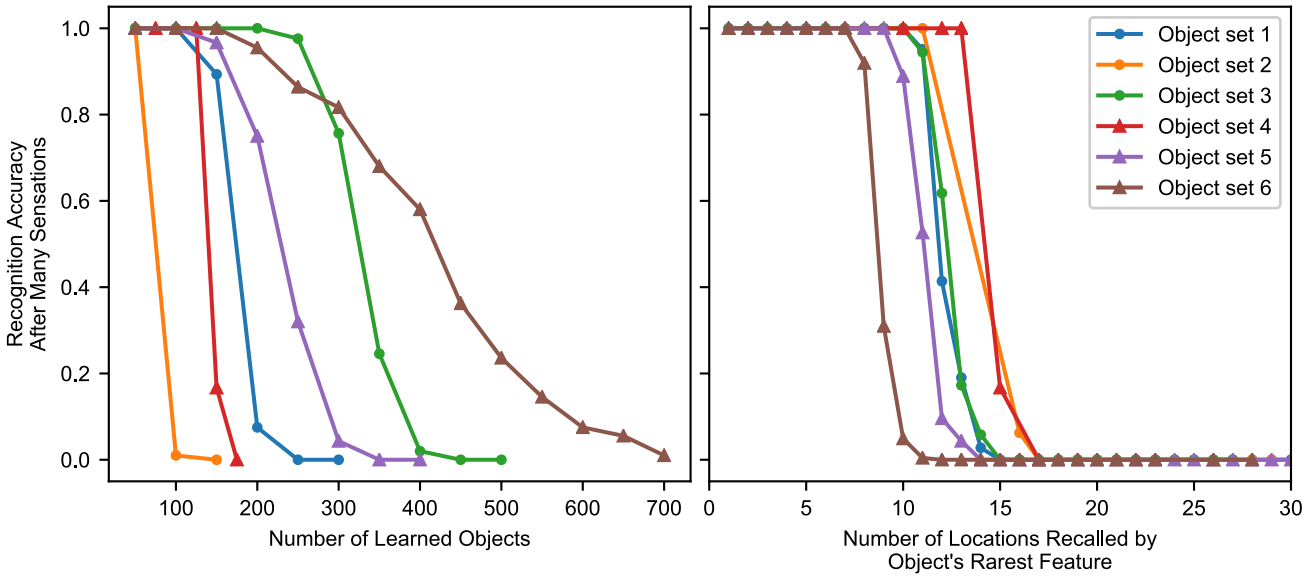


Figure S1. Varying the object statistics, the model's breaking point varies significantly relative to number of learned objects. The breaking point is much more consistent relative to the number of locations recalled by object features. In these charts we use a single model and test on 6 different distributions of objects. The model uses 10 modules with 10x10 cells per module. **(Left)** The network's capacity depends on the statistics of objects. The network's performance begins to break down after a certain number of objects, and this breaking point can vary by orders of magnitude with different object distributions. **(Right)** This breaking point varies significantly less when described in terms of "number of locations recalled by a sensation" rather than "number of objects learned". Using the same data from the first chart, for each object we measure the total number of occurrences of the object's rarest feature, and we plot recognition accuracy against this number. With each of these object distributions, the model reaches its breaking point when the number of recalled locations is within a small interval – conservatively, between 7 and 15. There is still some variation due to the statistics of the object's other features (not just its rarest feature), but the number of occurrences of the rarest feature provides a good first approximation for whether the network will recognize the object. **(Object descriptions)** Each object set had 100 unique features and 10 features per object, except where otherwise noted. The first three sets generate objects using the same strategy as all the other simulations, varying the parameters. The last three use different strategies. *Object Set 1:* baseline. *Object Set 2:* 40 unique features rather than 100. *Object Set 3:* 5 features per object rather than 10. *Object Set 4:* Every feature occurs the same number of times, +/- 1, rather than each object being randomly selected set of features with replacement. *Object Set 5:* Bimodal distribution of features, probabilistic. Divide features into two equal-sized pools, choose features from the second pool more often than features from the first. *Object Set 6:* Bimodal distribution of features, enforced structure. The features are divided equally into pools. Each object consists of one feature from the first pool and nine from the second.

The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding

Article *in* Frontiers in Computational Neuroscience · November 2017

DOI: 10.3389/fncom.2017.00111

3 authors, including:

Yuwei Cui
Numenta, Inc
26 PUBLICATIONS **282** CITATIONS
SEE PROFILE

Subutai Ahmad
Numenta
53 PUBLICATIONS **1,399** CITATIONS
SEE PROFILE

frontiers in Computational Neuroscience

ORIGINAL RESEARCH
published: 29 November 2017
doi: 10.3389/fncom.2017.00111

The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding

Yuwei Cui, Subutai Ahmad and Jeff Hawkins*

Numenta, Inc., Redwood City, CA, United States

Hierarchical temporal memory (HTM) provides a theoretical framework that models several key computational principles of the neocortex. In this paper, we analyze an important component of HTM, the HTM spatial pooler (SP). The SP models how neurons learn feedforward connections and form efficient representations of the input. It converts arbitrary binary input patterns into sparse distributed representations (SDRs) using a combination of competitive Hebbian learning rules and homeostatic excitability control. We describe a number of key properties of the SP, including fast adaptation to changing input statistics, improved noise robustness through learning, efficient use of cells, and robustness to cell death. In order to quantify these properties we develop a set of metrics that can be directly computed from the SP outputs. We show how the properties are met using these metrics and targeted artificial simulations. We then demonstrate the value of the SP in a complete end-to-end real-world HTM system. We discuss the relationship with neuroscience and previous studies of sparse coding. The HTM spatial pooler represents a neurally inspired algorithm for learning sparse representations from noisy data streams in an online fashion.

Keywords: hierarchical temporal memory, spatial pooler, sparse coding, competitive learning, Hebbian learning, online learning, sparse distributed representations

OPEN ACCESS

Edited by:
Anthony N. Burkitt,
University of Melbourne, Australia

Reviewed by:
Laurent U. Perrinet,
UMR7289 Institut de Neurosciences de la Timone (INT), France
Bailu Si,
University of Chinese Academy of Sciences (UCAS), China

****Correspondence:***
Subutai Ahmad
sahmad@numenta.com

Received: *27 June 2017*
Accepted: *15 November 2017*
Published: *29 November 2017*

Citation:
Cui Y, Ahmad S and Hawkins J (2017) The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding. Front. Comput. Neurosci. 11:111. doi: 10.3389/fncom.2017.00111

INTRODUCTION

Our brain continuously receives vast amounts of information about the external world through peripheral sensors that transform changes in light luminance, sound pressure, and skin deformations into millions of spike trains. Each cortical neuron has to make sense of a flood of time-varying inputs by forming synaptic connections to a subset of the presynaptic neurons. The collective activation pattern of populations of neurons contributes to our perception and behavior. A central problem in neuroscience is to understand how individual cortical neurons learn to respond to specific input spike patterns, and how a population of neurons collectively represents features of the inputs in a flexible, dynamic, yet robust way.

Hierarchical temporal memory (HTM) is a theoretical framework that models a number of structural and algorithmic properties of the neocortex (Hawkins et al., 2011). HTM networks can learn time-based sequences in a continuous online fashion using realistic neuron models that incorporate non-linear active dendrites (Antic et al., 2010; Major et al., 2013) with thousands of synapses (Hawkins and Ahmad, 2016). When applied to streaming data, HTM networks achieve state of the art performance on anomaly detection (Lavin and Ahmad, 2015; Ahmad et al., 2017) and sequence prediction tasks (Cui et al., 2016a).

The success of HTM relies on the use of sparse distributed representations (SDRs) (Ahmad and Hawkins, 2016). Such sparse codes represent a favorable compromise between local codes and dense codes (Földiák, 2002). It allows simultaneous representation of distinct items with little interference, while still maintaining a large representational capacity (Kanerva, 1988; Ahmad and Hawkins, 2015). Existence of SDRs have been documented in auditory, visual and somatosensory cortical areas (Vinje and Gallant, 2000; Weliky et al., 2003; Hromádka et al., 2008; Crochet et al., 2011). HTM spatial pooler (SP) is a key component of HTM networks that continuously encodes streams of sensory inputs into SDRs. Originally described in Hawkins et al. (2011), the term "spatial pooler" is used because input patterns that share a large number of co-active neurons (i.e., that are spatially similar) are grouped together into a common output representation. Recently there has been increasing interest in the mathematical properties of the HTM spatial pooler (Pietron et al., 2016; Mnatzaganian et al., 2017) and machine learning applications based on it (Thornton and Srbic, 2013; Ibrayev et al., 2016). In this paper, we explore several functional properties of the HTM spatial pooler that have not yet been systematically analyzed.

The HTM spatial pooler incorporates several computational principles of the cortex. It relies on competitive Hebbian learning (Hebb, 1949), homeostatic excitability control (Davis, 2006), topology of connections in sensory cortices (Udin and Fawcett, 1988; Kaas, 1997), and activity-dependent structural plasticity (Zito and Svoboda, 2002). The HTM spatial pooler is designed to achieve a set of computational properties that support further downstream computations with SDRs. These properties include (1) preserving topology of the input space by mapping similar inputs to similar outputs, (2) continuously adapting to changing statistics of the input stream, (3) forming fixed sparsity representations, (4) being robust to noise, and (5) being fault tolerant. As an integral component of HTM, the outputs of the SP can be easily recognized by downstream neurons and contribute to improved performance in an end-to-end HTM system.

The primary goal of this paper is to provide a thorough discussion of the computational properties of the HTM spatial pooler and demonstrate its value in end-to-end HTM systems. The paper is organized as follows. We first introduce the HTM spatial pooler algorithm. We discuss the computational properties in detail and then describe a set of metrics to quantify them. We demonstrate how these properties are satisfied, first using specific isolated simulations, and then in the context of an end-to-end HTM system. The main purpose of the paper is to study the role of the SP as demonstrated by applying it to HTM systems. In the discussion, we propose potential neural mechanisms and discuss the relationship to existing sparse coding techniques.

MODEL

The SP is a core component of HTM networks (**Figure 1A**). In an end-to-end HTM system, the SP transforms input patterns into SDRs in a continuous online fashion. The HTM temporal memory learns temporal sequences of these SDRs and makes predictions for future inputs (Cui et al., 2016a; Hawkins and Ahmad, 2016). A single layer in an HTM network is structured as a set of mini-columns, each with a set of cells (**Figure 1B**). The HTM neuron model incorporates dendritic properties of pyramidal cells in neocortex (Spruston, 2008), where proximal and distal dendritic segments on HTM neurons have different functions (**Figure 1C**) (Hawkins and Ahmad, 2016). Patterns detected on proximal dendrites lead to action potentials and define the classic receptive field of the neuron. Patterns recognized by a neuron's distal synapses act as predictions by depolarizing the cell without directly causing an action potential.

In HTM theory, different cells within a mini-column represent this feedforward input in different temporal contexts. The SP models synaptic growth in the proximal dendritic segments. Since cells in a mini-column share the same feedforward classical receptive field (Buxhoeveden, 2002), the SP models how this common receptive field is learned from the input. The SP output represents the activation of mini-columns in response to feedforward inputs. The HTM temporal memory models a cell's distal dendritic segments and learns transitions of SDRs by activating different sets of cells depending on the temporal context (Hawkins and Ahmad, 2016). The output of HTM temporal memory represents the activation of individual cells across all mini-columns.

The SP models local inhibition among neighboring mini-columns. This inhibition implements a k-winners-take-all computation (Majani et al., 1988; Makhzani and Frey, 2015). At any time, only a small fraction of the mini-columns with the most active inputs become active. Feedforward connections onto active cells are modified according to Hebbian learning rules at each time step. A homeostatic excitatory control mechanism operates on a slower time scale. The mechanism is called "boosting" in Hawkins et al. (2011), because it increases the relative excitability of mini-columns that are not active enough. Boosting encourages neurons with insufficient connections to become active and participate in representing the input.

Each SP mini-column forms synaptic connections to a population of input neurons. We assume that the input neurons are arranged topologically in an input space. We use $\mathbf{x}_j$ to denote the location of the jth input neuron and binary variable $\mathbf{z}_j$ to denote its activation state. The dimensionality of the input space depends on applications. For example, the input space is two-dimensional if the inputs are images and one-dimensional if the inputs are scalar numbers. A variety of encoders are available to deal with different data types (Purdy, 2016). The output neurons are also arranged topologically in a different space; we denote the location of the ith SP mini-column as $\mathbf{y}_i$.

We use HTM neuron models in the SP (**Figure 1B**). A complete description of the motivation and supporting evidence for this model can be found in Hawkins and Ahmad (2016). In this model, the learning rule is inspired by neuroscience studies of activity-dependent synaptogenesis (Zito and Svoboda, 2002). The synapses for the ith SP mini-column are located in a hypercube of the input space centered at $\mathbf{x}_i^c$ with an edge length of γ. Each SP mini-column has potential connections to a fraction of the inputs in this region. We call these "potential" connections because a

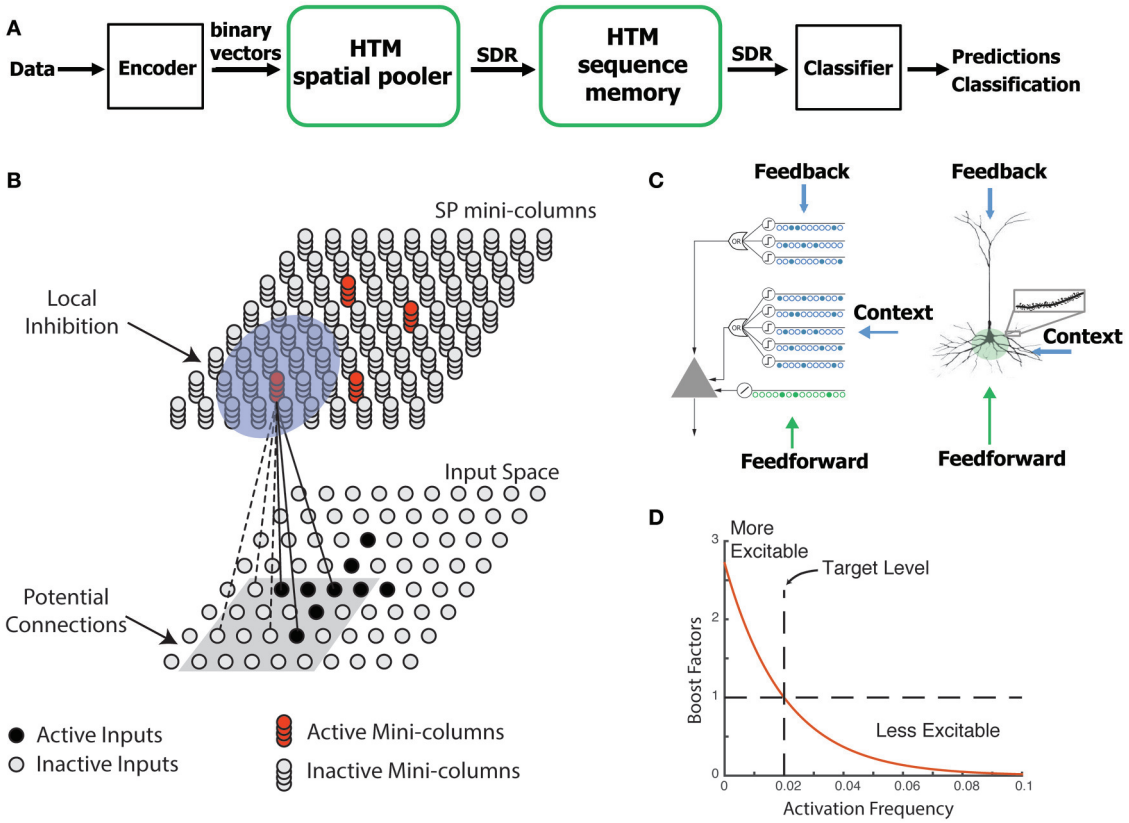


FIGURE 1 | HTM spatial pooler. **(A)** An end-to-end HTM system consists of an encoder, the HTM spatial pooler, the HTM temporal memory, and an SDR classifier. **(B)** The HTM spatial pooler converts inputs (bottom) to SDRs (top). Each SP mini-column forms synaptic connections to a subset of the input space (gray square, potential connections). A local inhibition mechanism ensures that a small fraction of the SP mini-columns that receive most of the inputs are active within the local inhibition radius (shaded blue circle). Synaptic permanences are adjusted according to the Hebbian rule: for each active SP mini-column, active inputs (black lines) are reinforced and inactive inputs (dashed lines) are punished. **(C)** An HTM neuron (left) has three distinct dendritic integration zones, corresponding to different parts of the dendritic tree of pyramidal neurons (right). The SP models the feedforward connections onto the proximal dendrite. **(D)** The excitability of a SP mini-column depends on its past activation frequency.

synapse is connected only if its synaptic permanence is above the connection threshold. The set of potential input connections for the *i*th mini-column is initialized as,

$$\Pi_i = \{ j \mid \mathrm{I}(\mathbf{x}_j; \mathbf{x}_i^c, \gamma) \; and \; Z_{ij} < p \} \tag{1}$$

$\mathrm{I}(\mathbf{x}_j; \mathbf{x}_i^c, \gamma)$ is an indicator function that returns one only if $\mathbf{x}_j$ is located within a hypercube centered at $\mathbf{x}_i^c$ with an edge length of γ. $Z_{ij} \sim U(0,\ 1)$ is a random number uniformly distributed in $[0, 1]$, p is the fraction of the inputs within the hypercube that are potential connections. The potential connections are initialized once and kept fixed during learning.

We model each synapse with a scalar permanence value and consider a synapse connected if its permanence value is above a connection threshold. We denote the set of connected synapses with a binary matrix $\mathbf{W}$,

$$W_{\mathbf{ij}} = \begin{cases} 1 \; if \; D_{ij} \geq \theta_c \\ 0 \; otherwise \end{cases} \tag{2}$$

where D_{ij} gives the synaptic permanence from the *j*th input to the *i*th SP mini-column. The synaptic permanences are scalar values between 0 and 1, which are initialized to be independent and identically distributed according to a uniform distribution between 0 and 1 for potential synapses.

$$D_{\mathbf{ij}} = \begin{cases} U(0,\ 1) \; if \; j \in \Pi_i \\ 0 \qquad otherwise \end{cases} \tag{3}$$

The connection threshold θ_c is set to be 0.5 for all experiments, such that initially 50% of the potential synapses are connected. Performance of the SP is not sensitive to the connection threshold parameter.

Neighboring SP mini-columns inhibit each other via a local inhibition mechanism. We define the neighborhood of the *ith* SP mini-column $\mathbf{y}_i$ as

$$N_i = \{ j \mid \left\| \mathbf{y_i} - \mathbf{y_j} \right\| < \phi, \; j \neq i \} \tag{4}$$

where $\left\| \mathbf{y}_i - \mathbf{y}_j \right\|$ is the Euclidean distance between the mini-column *i* and *j*. Since local inhibition occurs among neighboring mini-columns, the parameter ϕ controls the inhibition radius. Local inhibition is important when the input space has topology,

that is, when neighboring input neurons represent information from similar sub-regions of the input space. The inhibition radius is dynamically adjusted to ensure local inhibition affects mini-columns with inputs from the same region of the input space. That is, ϕ increases if the average receptive field size increases. Specifically, ϕ is determined by the product of the average connected input spans of all SP mini-columns and the number of mini-columns per input. If SP inputs and mini-columns have the same dimensionality, $\phi = \gamma$ initially. In practice we also deal with input spaces that have no natural topology, such as categorical information (Purdy, 2016). In this case there is no natural ordering of inputs and we use an infinitely large ϕ to implement global inhibition.

Given an input pattern $\mathbf{z}$, the activation of SP mini-columns is determined by first calculating the feedforward input to each mini-column, which we call the input overlap

$$o_i = b_i \sum_j W_{ij} z_j \tag{5}$$

where b_i is a positive boost factor that controls the excitability of each SP mini-column.

A SP mini-column becomes active if the feedforward input is above a stimulus threshold θ_{stim} and is among the top s percent of its neighborhood,

$$a_i = \begin{cases} 1 \; if \; o_i \geq \mathrm{Z}\,(V_i,\; 100 - s) \; and \; o_i \geq \theta_{stim} \\ 0 \; otherwise \end{cases} \tag{6}$$

We typically set θ_{stim} to be a small positive number to prevent mini-columns without sufficient input to become active. Z(X, p) is the percentile function that returns percentiles of the values in a data vector X for the percentages p in the interval [0, 100]. V_i is the overlap values for all neighboring mini-columns of the ith mini-column.

$$V_i = \{o_j | j \in N_i\} \tag{7}$$

s is the target activation density (we typically use $s = 2\%$). The activation rule (Equations 6–7) implements k-winners-take-all computation within a local neighborhood. It has been previously shown that such computation can be realized by integrate-and-fire neuron models with precise spike timings (Billaudelle and Ahmad, 2015). In this study, we use discrete time steps to speed up simulation.

The feedforward connections are learned using a Hebbian rule. For each active SP mini-column, we reinforce active input connections by increasing the synaptic permanence by p^+, and punish inactive connections by decreasing the synaptic permanence by p^-. The synaptic permanences are clipped at the boundaries of 0 and 1.

To update the boost factors, we compare the recent activity of each mini-column to the recent activity of its neighbors. We calculate the time-averaged activation level for the each mini-column over the last T inputs as

$$\bar{a}_i(t) = \frac{(T-1) * \bar{a}_i(t-1) + a_i(t)}{T} \tag{8}$$

where $a_i(t)$ is the current activity of the ith mini-column at time t. T controls how fast the boost factors are updated. Because the activity is sparse it requires many steps before we can get a meaningful estimate of the activation level. Typically we choose T to be 1,000. The time-averaged activation level in Equation (8) can be approximated by low-pass filtering of the voltage signal or intracellular calcium concentration. Similar calculations have been used in previous models of homeostatic synaptic plasticity (Clopath et al., 2010; Habenschuss et al., 2013).

The recent activity in the mini-column's neighborhood is calculated as

$$< \bar{a}_i(t) >= \frac{1}{|N_i|} \sum_{j \in N_i} \bar{a}_j(t) \tag{9}$$

Finally, the boost factor b_i is then updated based on the difference between $a_i\,(t)$ and $< \bar{a}_i(t) >$ as shown in **Figure 1D**.

$$b_i = e^{-\beta(\bar{a}_i(t) - <\bar{a}_i(t)>)} \tag{10}$$

Here, β is a positive parameter that controls the strength of the adaptation effect. The above boosting mechanism is inspired by studies of homeostatic regulation of neuronal excitability (see (Davis, 2006) for a review). The mechanism encourages efficient use of mini-columns by increasing the gain of mini-columns with sufficiently low average firing rate. The exact formula is not critical; we chose Equation (10) due to its simplicity.

RESULTS

Properties of the HTM Spatial Pooler

In this section, we describe a set of desirable properties for the HTM spatial pooler. These properties ensure flexible and robust representations of input streams with changing statistics, and are important for downstream neural computations.

The first property of the SP is to form fixed-sparsity representations of the input. To contribute to further neural computation, the outputs of the SP have to be recognized by downstream neurons. A cortical neuron recognizes presynaptic input patterns by initiating non-linear dendritic spikes (Major et al., 2013) or somatic action potentials (Bean, 2007), with thresholds depending on intrinsic cellular properties. It has been previously shown that recognition of presynaptic activation patterns is robust and reliable if the presynaptic inputs have a fixed level of sparsity (Ahmad and Hawkins, 2016). However, if the sparsity is highly variable, input patterns with high activation densities would be more likely to cause dendritic spikes or action potentials in downstream neurons, whereas patterns with low activation densities would be much harder to detect. This will contribute to high false positive error for high density patterns and false negative error for low density patterns. A fixed sparsity is desirable because it ensures all input patterns can be equally detected.

A second desirable property is that the system should utilize all available resources to learn optimal representations of the inputs. From an information theoretic perspective, neurons that are almost always active and neurons that do not respond to any

of the input patterns convey little information about the inputs. Given a limited number of neurons, it is preferable to ensure every neuron responds to a fraction of the inputs such that all neurons participate in representing the input space. The boosting mechanism in the SP (Equations 8–10) is designed to achieve this goal. We quantify this property using an entropy metric (see details below).

A third desirable property is that output representations should be robust to noise in the inputs. Real-world problems often deal with noisy data sources where sensor noise, data transmission errors, and inherent device limitations frequently result in inaccurate or missing data. In the brain, the responses of sensory neurons to a given stimulus can vary significantly (Tolhurst et al., 1983; Faisal et al., 2008; Masquelier, 2013; Cui et al., 2016b). It is important for the SP to have good noise robustness, such that the output representation is relatively insensitive to small changes in the input.

A fourth property is that the system should be flexible and able to adapt to changing input statistics. The cortex is highly flexible and plastic. Regions of the cortex can learn to represent different inputs in reaction to changes in the input data. If the statistics of the input data changes, the SP should quickly adapt to the new data by adjusting its synaptic connections. This property is particularly important for applications with continuous data streams that has fast-changing statistics (Cui et al., 2016a).

Finally, a fifth property is that the system should be fault tolerant. If part of the cortex is damaged, as might occur in stroke or traumatic brain injury, there is often an initial deficit in perceptual abilities and motor functions which is followed by substantial recovery that occurs in the weeks to months following injury (Nudo, 2013). It has also been documented that the receptive fields of sensory neurons reorganize following restricted lesions of afferent inputs, such as retinal lesions (Gilbert and Wiesel, 1992; Baker et al., 2005). The SP should continue to function in the event of system faults such as loss of input or output neurons in the network.

Spatial Pooler Metrics

In addition to gauging performance in end-to-end HTM systems, we would like to quantify the performance of SP as a standalone component. Since the SP is an unsupervised algorithm designed to achieve multiple properties, we describe several statistical metrics that can be directly calculated based on the inputs and outputs of the SP. Such metrics are particularly useful if configurations of the SP caused the end-to-end HTM system to have poor performance.

Metric 1: Sparseness

We define the population sparseness as

$$s^t = \frac{1}{N}\sum_{i=1}^{N} a_i^t \tag{11}$$

a_i^t is the activity of the ith mini-column at time step t, N is the number of SP mini-columns. This metric reflects the percentage of active neurons at each time step. Since we consider binary activations (Equation 6), the sparsity is straightforward to calculate. This metric has the same spirit as other population sparseness metrics for scalar value activations (Willmore and Tolhurst, 2001). We can quantify how well the SP achieves a fixed sparsity by looking at the standard deviation of the sparseness across time.

Metric 2: Entropy

Given a dataset of M inputs, the average activation frequency of each SP mini-column is

$$P(a_i) = \frac{1}{M}\sum_{t=1}^{M} a_i^t \tag{12}$$

The entropy of the ith SP mini-column is given by the binary entropy function

$$S_i = -P(a_i)\log_2 P(a_i) - (1 - P(a_i))\log_2(1 - P(a_i)) \tag{13}$$

If $P(a_i)$ equals zero or one, we set S_i to zero following convention. We define the entropy of the spatial pooler as

$$S = \sum\nolimits_{i=1}^{N} S_i \tag{14}$$

Since the average sparseness is almost constant in SP, the entropy is maximized when every SP mini-column has equal activation frequency. The SP will have low entropy if a small number of the SP mini-columns are active very frequently and the rest are inactive. Therefore, the entropy metric quantifies whether the SP efficiently utilizes all mini-columns.

Metric 3: Noise Robustness

We test noise robustness by measuring the sensitivity of the SP representation to varying amount of input noise. We denote a clean input and a noise contaminated input as $\mathbf{z}_i$ and $\mathbf{z}_i'(k)$, respectively, where k denotes the amount of noise added to the input. The corresponding SP outputs are denoted as $\mathbf{a}_i$ and $\mathbf{a}_i'(k)$, respectively. In our simulations, we randomly flip k percent of the active input bits to inactive, and flip the corresponding number of inactive input bits to active. This procedure randomizes inputs while maintaining constant input sparsity. We vary the amount of noise between 0 and 100%, and measure the fraction of shared active mini-columns in the SP output, averaged over a set of M inputs. The noise robustness index is defined as

$$R = \frac{1}{M}\sum_{i=1}^{M}\int_{k=0}^{1}\frac{\|\mathbf{a}_i \circ \mathbf{a}_i'(k)\|_0}{\|\mathbf{a}_i\|_0}dk \tag{15}$$

The L0-norm $\|\cdot\|_0$ gives the number of non-zero bits in a binary vector; the $\circ$ operator represents element-wise multiplication. Note that with binary vectors the L0-norm is identical to the L1-norm. The noise robustness index measures the area under the output overlap curve in **Figure 2C**. The fraction of shared active mini-columns start at 100% when noise is zero, and decreases toward 0 as the amount of noise increases. The noise robustness thus lies between 0 and 1.

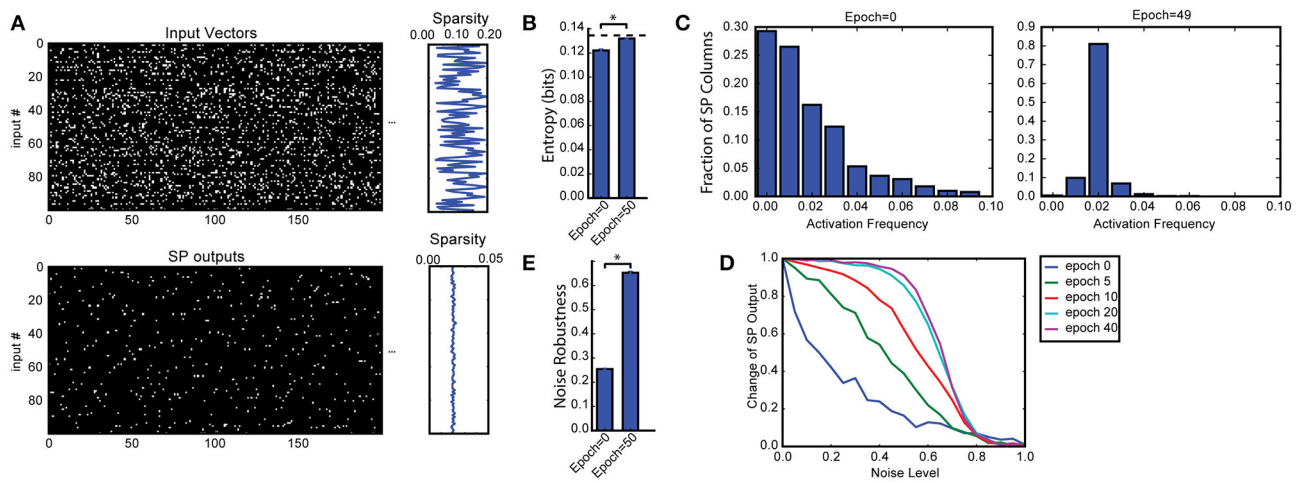


FIGURE 2 | Spatial pooler forms SDRs with fixed sparsity and good noise robustness. **(A)** We trained SP on a set of 100 randomly generated inputs (top). The input sparsity varies between 2 to 20%. The sparsity of the SP output lies close to 2% (bottom), despite the large variation of input sparsity. Only the first 200 dimensions of the input vectors are shown. **(B)** Entropy before and after learning, averaged across 10 repeated experiments ($^*p < 10^{-16}$, $n = 10$, paired t-test). The maximum possible entropy is shown as the black dashed line. **(C)** The distribution of activation frequency of SP mini-columns. Before learning (left), a significant fraction of the SP mini-columns (~30%) are not being used at all, while other mini-columns are active much more frequently. After learning (right), almost every mini-column is active for 2% of the time, suggesting every SP mini-column participate in representing the input. As a result, the entropy of the distribution is much higher. **(D,E)** Noise robustness of SP. We tested SP on noisy inputs during learning. **(D)** The change of the SP outputs is plotted as a function of the noise level. Before learning, a small amount of noise will lead to significant change in the SP output (blue), whereas after learning, there is almost no change in the SP output when 50% of the input bits changed. **(E)** Average noise robustness before and after learning ($^*p < 10^{-16}$, $n = 10$, paired t-test).

Metric 4: Stability

Since the SP is a continuously learning system, it is possible that the representation for a given input changes over time or becomes unstable. Instability without changes in the input statistics could negatively impact downstream processes. We measure stability by periodically disabling learning and presenting a fixed random subset of the input data. The stability index is the average fraction of active mini-columns that remained constant for each input.

Denote the SP output to the ith test input at the jth test point as $\mathbf{a}_i^j$, the stability index at test point j is given as

$$T(j) = \frac{1}{M}\sum_{i=1}^{M}\frac{\|\mathbf{a}_i^j \circ \mathbf{a}_i^{j-1}\|_0}{\|\mathbf{a}_i^{j-1}\|_0} \tag{16}$$

M is the number of inputs tested. The stability lies between 0 and 1, and equals 1 for a perfectly stable SP. Note that the superscript j is the index for test points instead of time steps in Equation (16). We compute the percentage overlap between the SP outputs to the same test inputs across consecutive test points. We train the SP on the entire set of training data between test points.

Simulation Details

We ran a number of different simulations (datasets described below). We used either a two-dimensional SP with 32 × 32 mini-columns for experiments with topology, or a dimensionless SP with 1,024 mini-columns for experiments without topology. The complete set of SP parameters is given in **Table 1**. The source code for all experiments are openly available at: https://github.com/numenta/htmpapers.

TABLE 1 | Parameters for the HTM spatial pooler.

Common parameters	Value
Activation density	2%
Connection threshold for synaptic permanence θ_c	0.5
Synaptic permanence increment p^+	0.1
Synaptic permanence decrement p^-	0.02
Boosting strength β	100
Activation frequency duty cycle T	1,000
Stimulus threshold θ_{stim}	1
Fraction of potential inputs p	1
PARAMETERS FOR THE ONE-DIMENSIONAL SPATIAL POOLER (NO TOPOLOGY)	
Column dimensions	1,024 × 1
Potential input radius γ	∞
PARAMETERS FOR THE TWO-DIMENSIONAL SPATIAL POOLER (WITH TOPOLOGY)	
Column dimensions	32 × 32
Potential input radius γ	5

We presented each dataset in a streaming online fashion. Each dataset is presented to the SP in one or more epochs. We define an epoch as a single pass through the entire dataset in random order. Note that this definition of epoch is different from batch training paradigm because the SP receives one input pattern at a time and does not maintain any buffer of the entire dataset. We measured SP metrics between epochs on a random subset of the input data with learning turned off. In practice, the metrics could also be monitored continuously during learning.

We used the following datasets:

Random Sparse Inputs

In this experiment we created a set of 100 random inputs with varying sparsity levels. Each input is a 32 × 32 image where a small fraction of the bits are active. The fraction of active inputs is uniformly chosen between 2 to 20%. This dataset employs a SP with topology and is used in **Figures 2**, **3**.

Random Bars and Crosses

The random bar dataset consists of 100 pairs of random bars. Each random bar pair stimuli is a 10 × 10 image, with a horizontal bar and a vertical bar placed at random locations. The bars have a length of five pixels. Each random cross stimuli is a 10 × 10 image with a single cross. The random cross dataset consists of 100 cross patterns at random locations, where each cross consists of a horizontal bar and a vertical bar that intersect at the center. Each bar has a length of five pixels. This dataset is used in **Figures 4A,B**.

MNIST

We trained a SP without topology on the MNIST database of handwritten digits (Lecun et al., 1998). We present the full training set of 60,000 examples in a single epoch to the SP. We visually examined the receptive field structures of a subset of randomly selected SP mini-columns after training. This dataset is used in **Figure 4C**.

Fault Tolerance with Topology

For the fault tolerance experiment (**Figure 5**), we used images of random bar sets as input. The input space has dimensionality of 32 × 32 and each input contains six randomly located horizontal or vertical bars. Each bar has a length of 7. We used an SP with a two-dimensional topology and 32 × 32 mini-columns. We first trained the intact SP on the random bars input until it stabilized (after 18,000 inputs). We then tested two different types of trauma: simulated stroke or simulated input lesion. During the simulated stroke experiment, we permanently eliminated 121 SP mini-columns that lie in an 11 × 11 region at the center of the receptive field. For the simulated input lesion experiment, we did not change the SP during the trauma. Instead, we permanently blocked the center portion of the input space (121 inputs that lie in an 11 × 11 region at the center of the input space). For both experiments, we monitored the recovery of the SP for another 42,000 steps.

NYC Taxi Passenger Count Prediction

In addition to the above artificial datasets, we also tested the SP in an end-to-end real-world HTM system. We chose the problem of demand prediction for New York City taxis. The dataset is publicly available via the New York City Metropolitan Authority. Full details are described in our previous paper (Cui et al., 2016a). As described in **Figure 1**, the input data stream is first converted to binary representations using a set of encoders (Purdy, 2016). The SP takes the output of the encoders as input and forms SDRs. The HTM sequence memory then learns sequences of SDRs and represents sequences with a sparse temporal code. Finally, we use a single layer feedforward classification network to map outputs of HTM sequence memory into real-time predictions for future inputs. The task is to model a continuous stream within the context of a real-time application. As such the SP, temporal memory, and classifier all learn continuously. It is important for the SP to output robust and efficient representations in order for the downstream components to learn.

Following Cui et al. (2016a), we aggregated the passenger counts in New York City taxi rides at 30-min intervals. We encoded the current passenger count, time of day and day of week

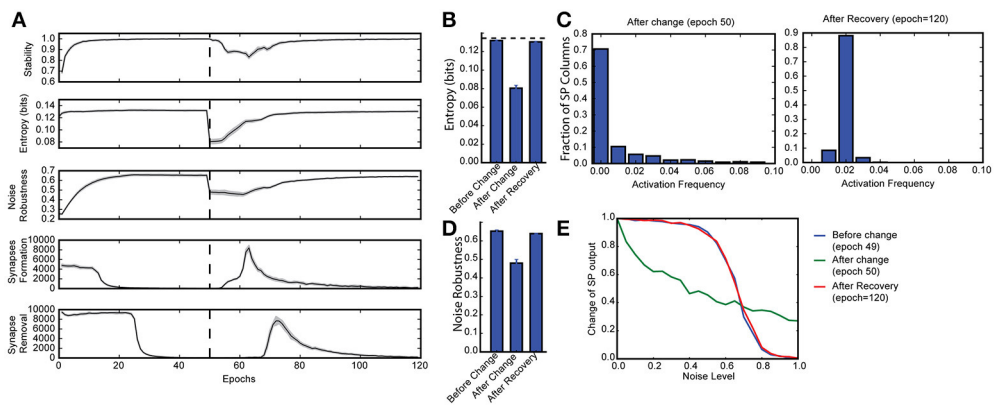


FIGURE 3 | Continuous learning with HTM spatial pooler. SP continuously adapts to the statistics of the input data. SP is trained on a set of random inputs (described in **Figure 2**) until it stabilizes. We then switch to a new set of inputs (black dashed line) and monitor the continuous adaptation of SP to the new dataset. (**A)** Statistical metrics on SP during continuous learning: top: stability, second row: entropy; third row: noise robustness; fourth row: formation of new synapses; fifth row: removal of synapses. **(B)** The entropy decreases right after the change of the input dataset (epoch = 50), and recovers completely after the SP is trained on the new dataset for long enough time (epoch = 120). The black dashed line showed the theoretical limit for entropy given the sparsity constraint. **(C)** Distribution of activation frequency of SP mini-columns right after the change in dataset (left) and after recovery (right). **(D)** Noise robustness before change (epoch = 49), right after change (epoch = 50), and after recovery (epoch = 120). **(E)** The noise robustness decreases right after the change of the input dataset (blue vs. green), and recovers completely after the SP is sufficiently trained on the new dataset (red).

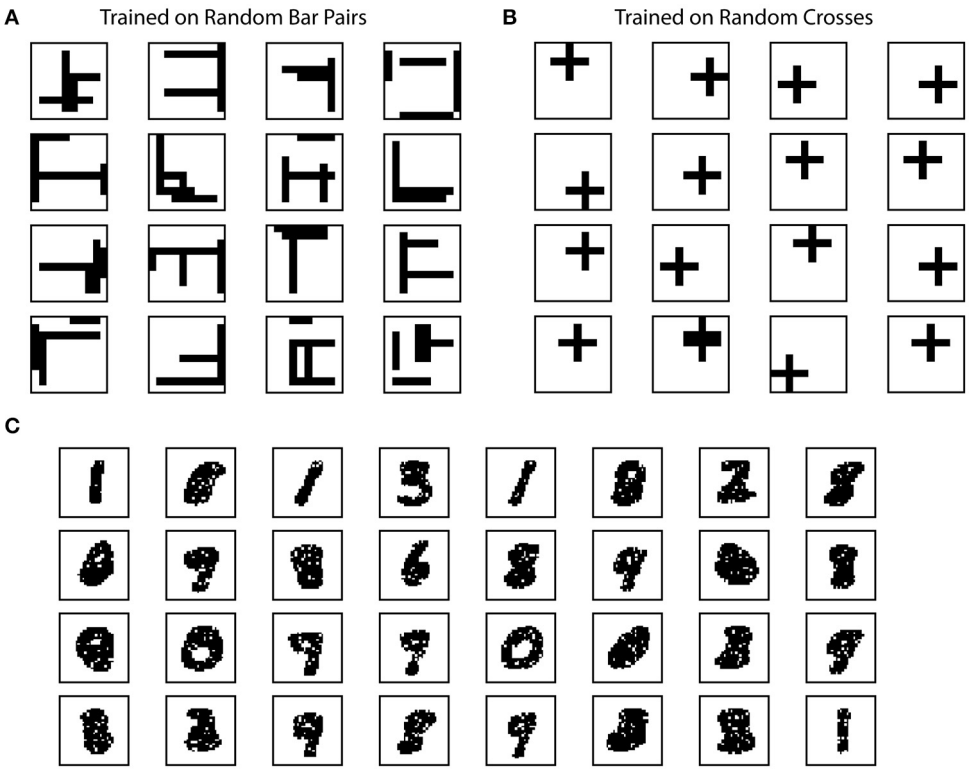


FIGURE 4 | Example receptive fields of SP. The receptive fields of SP mini-columns capture statistics of the input data. We define receptive field as the set of inputs that are connected to a mini-column. **(A)** Example SP Receptive fields trained on random bar pairs. **(B)** Example SP receptive fields trained on random crosses. **(C)** Example SP receptive fields trained on MNIST dataset.

into binary vectors using scalar and date-time encoders (Purdy, 2016). A SP with global inhibition was trained continuously on the outputs of encoders and provided input to the HTM sequence memory (Cui et al., 2016a). To evaluate the role of learning and boosting in SP, we compared the prediction accuracy in three scenarios (1) SP without learning or boosting, (2) SP with learning but not boosting, and (3) SP with both learning and boosting.

Simulation Results

We first discuss results on the Random Sparse Inputs dataset with respect to the metrics (**Figures 2**, **3**). The input patterns are presented repeatedly to the HTM spatial pooler in a streaming fashion. In our simulation, the population sparsity of the SP is always close to the target level of 2%, even though the input sparsity varies widely in the range of 2–20% (**Figure 2A**). This is an inherent property of the network due to the use of local k-winners-take-all activation rules.

We measured the average entropy across all mini-columns (see section Spatial Pooler Metrics). Since the overall activation sparsity is fixed in our network, the entropy is maximized if all mini-columns have the same activation probability. In this experiment, the entropy increases from 0.1221 ± 0.0013 bits/mini-column to 0.1320 ± 0.0007 bits/mini-column with training. The difference is highly significant across repeated experiments with different set of random inputs ($p < 10^{-8}$, $n = 10$, paired t-test). As a reference, the maximum possible entropy is 0.1345 bits/mini-column with the same sparsity levels. The increase of entropy is due to efficient use of all mini-columns. Before learning, a significant fraction of the mini-columns (∼30%) were not active for any of the input, whereas a small fraction of the mini-columns were much more active than others. After learning, almost every mini-column was active for 2% of the time.

Figures 2D,E demonstrate noise robustness as a function of SP learning. Before learning, a small change in the input will cause a large change in the SP output, suggesting high noise sensitivity (**Figure 2D**, blue). After learning, the noise robustness gradually improves. After 40 repetitions of the dataset, there is no change on the SP output even if 40% of the active input bits are changed. The average noise robustness index (Equation 15) correspondingly improves from 0.254 ± 0.004 to 0.652 ± 0.007 (**Figure 2E**, $p < 10^{-16}$, $n = 10$, paired t-test). The improved noise robustness is due to the Hebbian learning rules. A set of SP mini-columns forms reliable connections to active input neurons during learning. The same set of SP mini-columns can be activated even if some of the input neurons are affected by noise after learning.

To test whether the SP can adapt to changing inputs, we train the SP until it stabilizes on one set of inputs. We then

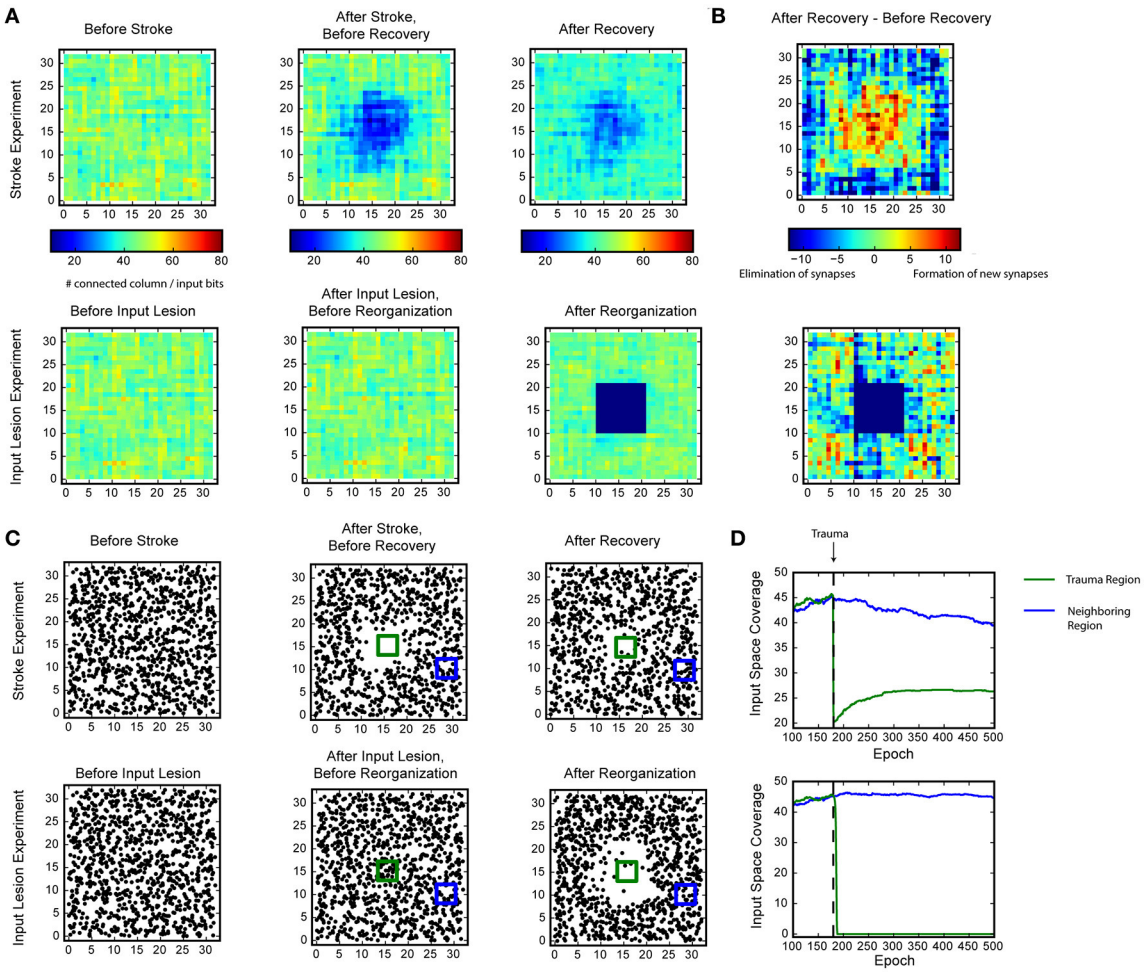


FIGURE 5 | Recovery of HTM spatial pooler after damage and input lesion. During the simulated stroke, a fraction of SP mini-columns that are connected to the center region of the input space is killed. During the simulated retinal lesion, the center portion of the input space is blocked while the spatial pooler and its feedforward inputs are kept intact. **(A)** The number of SP mini-columns connected to each input bits before trauma (left), right after trauma (middle), and after recovery (right). The simulated stroke experiment is shown at the top and the simulated retinal lesion experiment is shown at the bottom. **(B)** Growth and elimination of synapses during the recovery process for the simulated stroke (top) and retinal lesion (bottom) experiment. **(C)** Receptive field centers of all SP mini-columns before trauma (left), right after trauma (middle), and after recovery (right). **(D)** Number of mini-columns connected to the center region [green square in **(C)** in this figure] and a neighboring region (blue square) during the recovery process. The recovery is very fast for the retinal lesion experiment (bottom), and slower for the simulated stroke experiment (top).

present a completely different input dataset (**Figure 3A**). Right after we switch to a new dataset, the entropy and noise robustness drops sharply (**Figures 3B,D**). At this point a large fraction of the SP mini-columns are not responsive to any input in the new dataset (**Figure 3C**, left). Once learning resumes the SP quickly adapts to the new input dataset and the performance metrics recover back to the levels before the change (**Figures 3B,D**). The SP adapts to the new dataset by first forming many more new synapses (**Figure 3A**, fourth row) and then pruning unnecessary connections later (**Figure 3A**, fifth row). This shows that the SP can learn a new dataset even after learning has stabilized. Note that due to the boosting rule, which encourages reuse of all mini-columns, the original dataset will be forgotten.

The SP achieves these properties by continuously adapting feedforward connections to the input data. To illustrate how receptive field structures of SP are shaped by the input data, we trained the SP on the Random Bars dataset. In this experiment, the training data consists of either pairs of randomly generated bars, or a single cross at a random location.

We plot example receptive fields from a random subset of SP mini-columns in **Figure 4**. For this dataset the receptive field typically contains horizontal and vertical bar structures at random locations (**Figure 4A**). Each SP mini-column responds to more than one bar in order to achieve the target activation frequency of 2%. When trained on the random cross data stream, the resulting receptive field consists of cross structures that resemble statistics of the inputs (**Figure 4B**). The results are largely unaffected by

the number of mini-columns in the network. The resulting receptive field depends on the ratio of synaptic permanence decrement and increment (p^-/p^+). If we increase p^- to 0.05, most of the receptive field contains single bar segment when trained on the random bar pairs dataset (data not shown).

We trained the SP on the MNIST dataset. In this case the receptive field contains digit-like structures (**Figure 4C**). Although some receptive fields clearly detect single digits, others are responsive to multiple digits. This is because individual SP mini-columns do not behave like "grandmother cells"—they are not meant to detect single instances of the inputs. Instead a single input is collectively encoded by a set of SP mini-columns. When trained on complex natural datasets, we expect to see a diversity of receptive structures within the SP, which may not resemble specific input instances.

We also tested the SP's ability to represent mixed or ambiguous inputs. After training the SP on the datasets used in **Figure 4**, we compared the average distance between pairs of random inputs vs. a mixture of the pair. In general, the SP representation for a merged input has a greater similarity to the SP representation of the original inputs than to a random input. As an example, for the random crosses dataset, the average overlap between the representation of two random inputs is about 6.5. If we create a merged input the overlap between its SP representation and the representation of the individual inputs jumps to about 42. Thus, the SP representation of merged inputs retains significant similarity to the representation of the original inputs.

Fault Tolerance

We evaluated whether the HTM spatial pooler has the ability to recover from lesion of the afferent inputs (input lesion experiment) or damage to a subset of the SP mini-columns (stroke experiment).

In the stroke experiment, the center portion of the input space becomes much less represented right after the trauma because the corresponding SP mini-columns are eliminated. The network partially recovers from the trauma after a few hundred epochs, with each epoch consisting of 100 inputs. During the recovery process, SP mini-columns near the trauma region shift their receptive field toward the trauma region and start to represent stimuli near the center (**Figure 5C**, Supplementary video 1). The network forms many more new synapses in the center, which is accompanied by loss of synapses in the non-trauma region (**Figure 5B**, top). There is a clear recovery on the coverage of the trauma region (**Figure 5D**, bottom).

In the input lesion experiment, the center portion of the input space is blocked. Since there is no change on the SP mini-columns or the associated synaptic connections, there is no immediate change on the input space coverage (**Figure 5A**, bottom) or the receptive field center distributions (**Figure 5C**, bottom). However, the SP mini-columns quickly reorganize their receptive field within a few epochs. The SP mini-columns that respond to the center inputs starts to respond to inputs on the surrounding, non-damaged areas (**Figure 5C**, Supplementary video 2). Almost all the connections to the lesion region are lost after the reorganization (**Figure 5B**, bottom).

These demonstrate the fault tolerance and flexibility of the SP. The fixed sparsity and the homeostasis excitability control mechanism of the SP ensure that the input space is efficiently represented by all SP (undamaged) mini-columns. It is interesting to note that the different recovery speeds from the two simulations coincide with experimental studies. It has been reported that after focal binocular retinal lesions, the receptive field sizes increases within a few minutes for cortical neurons that lie near the edge of the retinal scotoma (Gilbert and Wiesel, 1992). In contrast, if part of the cortex is damaged, the recovery is partial and occurs on a much slower time scale (Nudo, 2013).

The Spatial Pooler in a Real-World Streaming Analytics Task

In this section we evaluate the role of the SP in an end-to-end real-world HTM system. We consider the problem of real-time prediction of the number of taxi passenger in New York City (**Figure 6A**, see section Methods). We have previously shown that HTM systems with a fixed pre-trained SP achieves state-of-the-art performance on this task (Cui et al., 2016a). Here we consider the role of learning in the SP and evaluate three scenarios: using a randomly initialized SP without learning, allowing SP learning but without boosting (boost strength set to 0), and using a SP with both continuous learning and boosting (boost strength set to 100).

We fed the inputs in a single pass to mimic the scenario of real-time online prediction. The time-averaged prediction error for the three cases is plotted as a function of training time (**Figure 6B**). At the beginning of learning, the prediction error rapidly decreases, representing the initial learning phase of the system. The occasional increases in error reflect real world changes that correspond to events and holidays (e.g., Thanksgiving, Christmas, New York City Marathon, etc.).

The SP with both learning and boosting achieves the best performance throughout the prediction task. The SP with learning but without boosting is roughly comparable to a random static SP. This suggests the importance of both continuous Hebbian learning and homeostatic excitability control. The difference in performance can be understood by observing the distribution of activation frequency across SP mini-columns. Without the homeostatic excitability control, a large fraction of the SP mini-columns are not being used at all (**Figure 6C**). It is more error-prone for the HTM sequence memory to learn transitions of such ill-behaved SDRs.

DISCUSSION AND CONCLUSIONS

In this paper, we described properties of the HTM spatial pooler, a neurally inspired algorithm for learning SDRs online. Inspired by computational principles of the neocortex, the goal of the HTM spatial pooler is to create SDRs and support essential neural computations such as sequence learning and memory. The model satisfies a set of important properties, including tight control of output sparsity, efficient use of mini-columns, preserving similarity among inputs, noise robustness, fault tolerance, and

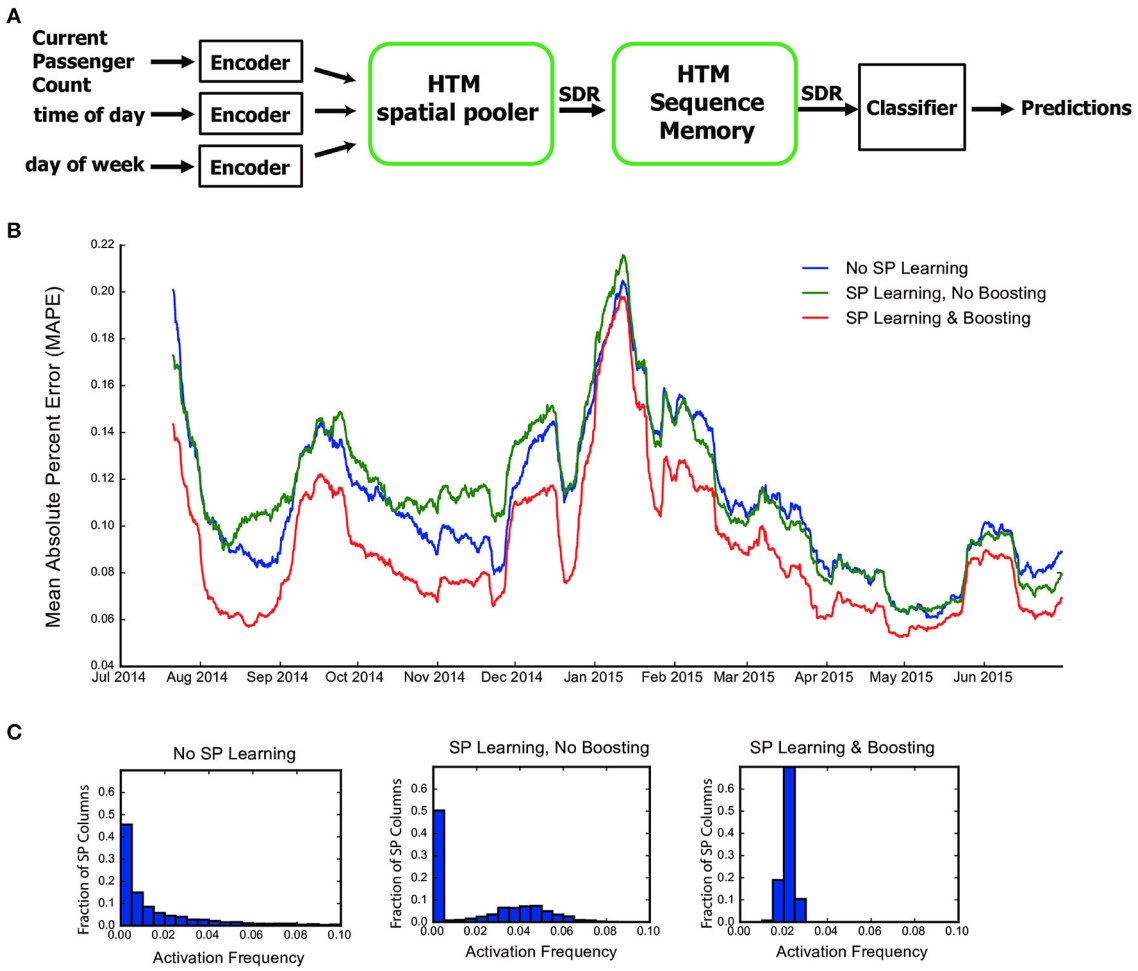


FIGURE 6 | The role of the HTM spatial pooler in an online prediction task. **(A)** A complete HTM system is used for predicting the NYC taxi passenger count. **(B)** Prediction error with an untrained random SP (blue), a SP with continuous learning but without boosting (green), and a SP with both continuous learning and boosting (red). **(C)** Distribution of activation frequency of SP mini-columns. A large fraction of SP mini-columns are not being used for an untrained SP or for a SP without boosting. In contrast, almost all mini-columns are active for about 2% of the time when boosting is enabled.

fast adaptation to changes. These properties are achieved using competitive Hebbian learning rules and homeostatic excitability control mechanisms. We demonstrate the effectiveness of SP in an end-to-end HTM system on the task of streaming data prediction. The HTM spatial pooler leads to a flexible sparse coding scheme that can be used in practical machine learning applications.

Relationship with Other Sparse Coding Techniques

The SP learns SDRs for inputs. It is related to the broad class of sparse coding techniques, which uses activation of a small set of neurons to encode each item. One theory of sparse coding suggests that sparse activations in sensory cortices reduce energy consumption of the brain while preserving most of the information (Földiák, 2002; Olshausen and Field, 2004). Early studies of sparse coding explicitly optimize a cost function that combines low reconstruction error and high sparseness (Olshausen and Field, 1996a, 1997). When applied to natural images, these techniques lead to receptive fields that resemble those of V1 neurons (Olshausen and Field, 1996a; Lee et al., 2006), suggesting that the functionality of early sensory neurons can be explained by the sparse coding framework. Sparse coding has been implemented previously with biologically plausible local learning rules. Földiák showed that a neural network could learn a sparse code using Hebbian forward connections combined with a local threshold control mechanism (Földiák, 1990). It has been recently shown that such learning rules can be derived analytically (Hu et al., 2014).

Many of the properties we analyzed in this paper have also been discussed in previous studies of sparse coding. It has been shown that sparse representations are naturally robust to noise and can be used for robust speech recognition (Sivaram et al., 2010; Gemmeke et al., 2011), robust face recognition (Wright et al., 2009) and super resolution image reconstruction (Yang et al., 2010). Online sparse coding and dictionary learning

techniques have been proposed in previous studies in order to handle dynamic datasets (Mairal et al., 2010). It is known that representations learned from traditional sparse coding techniques have low entropy, as the probability distribution of activity of an output unit is peaked around zero with heavy tails (Olshausen and Field, 1996b). In this study we found that although the entropy is low compared to dense representations, it increases with training in the HTM spatial pooler. This is because the homeostatic excitability control mechanism encourages neurons in the SP to have similar activation frequencies, thus increasing the representational power of the network.

Most previous studies propose the goal of sparse coding is to avoid information loss, reduce energy consumption, and form associative memory with minimum cross talk (Olshausen and Field, 2004). A commonly used criterion is how well one can reconstruct the inputs given the sparse activations and a set of learned basis vectors. Although these studies explain receptive structure in primary visual cortex and lead to practical machine learning algorithms for feature selection (Gui et al., 2016) and data compression (Pati et al., 2015), the purpose of neural computation is more than preserving information. In this paper, we take a different perspective and ask how computational properties of the HTM spatial pooler contribute to downstream cortical processing in the context of HTM systems. Instead of reconstruction error, we define the performance of SP in terms of a set of properties, including population entropy, noise robustness, stability, and fault tolerance. It is important to perform a multi-dimensional assessment of the SP in order to ensure that it forms robust SDRs that capture topological properties of the input space. We demonstrated that the HTM spatial pooler achieves these properties and that these properties contribute to improved performance in an end-to-end system. It is important to note that no single metric is sufficient to ensure SP is behaving properly. For example, one can achieve good noise robustness by always using a small set of SP mini-columns, but that will give bad entropy. It is easy to achieve high entropy by using a random output at each time step, but that will cause bad stability. It is important to consider all the metrics together. As a result, the learning algorithm of SP cannot be easily derived by optimizing a single objective function.

The Hebbian learning rules of HTM spatial pooler resemble many previous sparse coding algorithms (Földiák, 1990; Zylberberg et al., 2011; Hu et al., 2014) and associative memory models (Willshaw et al., 1969; Hecht-Nielsen, 1990; Bibbig et al., 1995). There are several differences. First, we include homeostatic excitability control as a gain modulation mechanism. The role of homeostasis is to make sure that the distribution of neural activity is homogeneous. It has been previously proposed that homeostasis is crucial in providing an efficient solution when learning sparse representations (Perrinet, 2010). Some models of synaptic plasticity do include homeostatic components in the learning rule that control the amount of synaptic weight change (Clopath et al., 2010; Habenschuss et al., 2013). The homeostatic excitability regulation mechanism in the SP achieves a similar effect without directly affecting the synapse modification process. Second, we simulate a local inhibition circuit that implements k-winners-take-all computation to have tight control over the output sparseness (**Figure 7A**). This is important when SP activations are used by downstream neurons with dendrites that have threshold non-linearities. We do not explicitly model a continuous time network of excitatory and inhibitory neurons as in balanced networks (Hansel and Mato, 2013; Denève and Machens, 2016) but have a similar goal of maintaining a tight range of sparsity. Finally, we use binary synapses and learning via synaptogenesis (Zito and Svoboda, 2002). The use of binary synapses can dramatically speed up the computation. Overall the HTM spatial pooler algorithm is a suitable candidate for learning sparse representations online from streaming data.

Potential Neural Mechanisms of Spatial Pooler

A layer in a HTM system contains a set of mini-columns. Each mini-column contains cells with the same feedforward receptive field. The mini-column hypothesis has been proposed for several decades (Mountcastle, 1997; Buxhoeveden, 2002), but the utility of mini-columns remains controversial due to a lack of theoretical benefit (Horton and Adams, 2005). According to HTM theory cells within the same mini-column have the same feedforward connections but different lateral connections, thus representing the same feedforward input in different temporal contexts (Hawkins and Ahmad, 2016). We propose the SP models feedforward receptive field learning at the mini-column level. Experimental studies have shown that neurons within the same mini-column have almost identical receptive field locations, sizes and shapes, whereas RFs of neurons

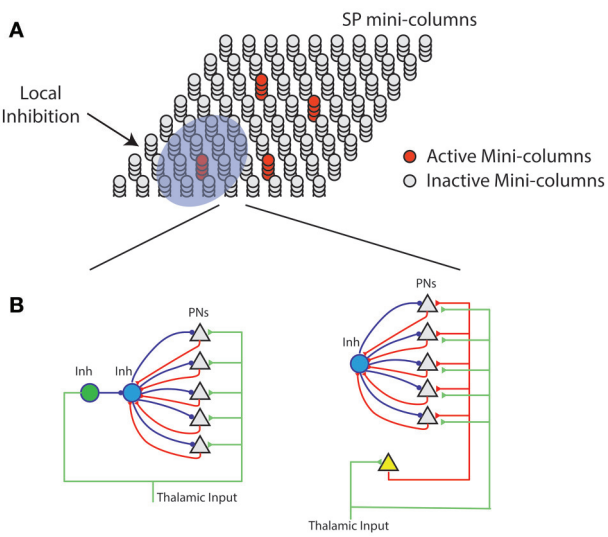


FIGURE 7 | Neural mechanism of HTM spatial pooler. **(A)** Spatial pooler requires local inhibition across mini-columns to ensure that a small fraction of the mini-columns are active at any time. **(B)** Potential mechanisms to ensure neurons within the same mini-column share the same feedforward receptive field. **(**Left) The green inhibitory neuron controls the receptive fields of excitatory pyramidal neurons (gray triangles) through a dis-inhibition circuit. (right) A single (or small number of) excitatory neurons (yellow) controls the receptive field of excitatory neurons. In both cases, PNs indirectly inhibit other PNs in the same mini-column through the within mini-column inhibition (blue).

in neighboring mini-columns can differ significantly (Jones, 2000). This variability cannot be explained by the difference in feedforward inputs, because the extent of arborization of single thalamic afferent fibers, which can be as much as 900 μm in cats (Jones, 2000), is significantly more extensive than the dimensions of minicolumns, which typically have diameters around 20–60 μm (Favorov and Kelly, 1996; Jones, 2000; Buxhoeveden, 2002). It requires dedicated circuitry mechanism to ensure that cells in the same mini-column acquire the same receptive field.

We propose two possible neural circuit mechanisms for the SP and discuss their anatomical support. In the first proposal (**Figure 7B**, left), the feedforward thalamic inputs innervate both excitatory pyramidal neurons as well as an inhibitory neuron (*green*). This inhibitory neuron can indirectly activate the pyramidal neurons through a disynaptic dis-inhibition circuit. It acts as a "teacher" cell that guides the receptive field formation of excitatory neurons. There are many distinct classes of inhibitory neurons in the cortex. Some classes, such as bipolar cells and double bouquet cells exclusively innervate cells within a cortical mini-column (Markram et al., 2004; Wonders and Anderson, 2006). It is well-documented that feedforward thalamacortical input strongly activates specific subtypes of inhibitory neurons (Gibson et al., 1999; Porter et al., 2001; Swadlow, 2002; Kremkow et al., 2016). It is possible these inhibitory neurons participate in defining and maintaining the feedforward receptive field of cortical mini-columns.

In the second proposal, a single excitatory cell receives thalamic inputs and innervates all excitatory cells in a mini-column. This excitatory neuron guides the receptive field formation of other excitatory cells in the mini-column. A similar circuit has been observed during early development. Subplate neurons, a transient population of neurons, receive synaptic inputs from thalamic axons, establishing a temporary link between thalamic axons and their final targets in layer IV (Friauf et al., 1990; Ghosh and Shatz, 1992; Kanold et al., 2003). It remains to be tested whether a similar circuit exists in adult brain.

The SP relies on several other neural mechanisms. The learning rule is based on competitive Hebbian learning. Such learning can be achieved in the brain via synaptic plasticity rules such as long-term potentiation (Teyler and DiScenna, 1987), long-term depression (Ito, 1989), or spike-time dependent plasticity (Song et al., 2000). Homeostatic excitability control mechanisms, analogous to the SP's boosting rule, have been observed in cortical neurons (Davis, 2006). Finally, the *k*-winners-take-all computation in the SP can be implemented using leaky integrate-and-fire neuron models (Billaudelle and Ahmad, 2015).

AUTHOR CONTRIBUTIONS

YC, SA and JH: wrote the paper together and conceived the experiments; YC and SA: formulated the mathematical metrics of spatial pooler; and YC: performed the experiments.

FUNDING

Numenta is a privately held company. Its funding sources are independent investors and venture capitalists.

ACKNOWLEDGMENTS

We thank the editor and the reviewers for their constructive comments and suggestions. We thank Mirko Klukas for suggestions on the entropy metric. We also thank the NuPIC open source community (numenta.org) for continuous support and enthusiasm about HTM.

SUPPLEMENTARY MATERIAL

The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/fncom.2017.00111/full#supplementary-material

Video 1 | During the recovery process of the stroke experiment, SP mini-columns near the trauma region shift their receptive field toward the trauma region and start to represent stimuli near the center.

Video 2 | In the input lesion experiment, the SP mini-columns that respond to the center inputs start to respond to inputs on the surrounding, non-damaged areas.

REFERENCES

Ahmad, S., and Hawkins, J. (2015). Properties of sparse distributed representations and their application to hierarchical temporal memory. *arXiv* arXiv:1503.07469.

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

Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* 262, 134–147. doi: 10.1016/j.neucom.2017.04.070

Antic, S. D., Zhou, W. L., Moore, A. R., Short, S. M., and Ikonomu, K. D. (2010). The decade of the dendritic NMDA spike. *J. Neurosci. Res.* 88, 2991–3001. doi: 10.1002/jnr.22444

Baker, C. I., Peli, E., Knouf, N., and Kanwisher, N. G. (2005). Reorganization of visual processing in macular degeneration. *J. Neurosci.* 25, 614–618. doi: 10.1523/JNEUROSCI.3476-04.2005

Bean, B. P. (2007). The action potential in mammalian central neurons. *Nat. Rev. Neurosci.* 8, 451–465. doi: 10.1038/nrn2148

Bibbig, A., Wennekers, T., and Palm, G. (1995). A neural network model of the cortico-hippocampal interplay and the representation of contexts. *Behav. Brain Res.* 66, 169–175. doi: 10.1016/0166-4328(94)00137-5

Billaudelle, S., and Ahmad, S. (2015). Porting HTM models to the Heidelberg neuromorphic computing platform. *arXiv* arXiv:1505.02142 [q-NC].

Buxhoeveden, D. P. (2002). The minicolumn hypothesis in neuroscience. *Brain* 125, 935–951. doi: 10.1093/brain/awf110

Clopath, C., Büsing, L., Vasilaki, E., and Gerstner, W. (2010). Connectivity reflects coding: a model of voltage-based STDP with homeostasis. *Nat. Neurosci.* 13, 344–352. doi: 10.1038/nn.2479

Crochet, S., Poulet, J. F. A., Kremer, Y., and Petersen, C. C. H. (2011). Synaptic mechanisms underlying sparse coding of active touch. *Neuron* 69, 1160–1175. doi: 10.1016/j.neuron.2011.02.022

Cui, Y., Ahmad, S., and Hawkins, J. (2016a). Continuous online sequence learning with an unsupervised neural network model. *Neural Comput.* 28, 2474–2504. doi: 10.1162/NECO_a_00893

Cui, Y., Liu, L., McFarland, J., Pack, C., and Butts, D. (2016b). Inferring Cortical variability from local field potentials. *J. Neurosci.* 36, 4121–4135. doi: 10.1523/JNEUROSCI.2502-15.2016

Davis, G. W. (2006). Homeostatic control of neural activity: from phenomenology to molecular design. *Annu. Rev. Neurosci.* 29, 307–323. doi: 10.1146/annurev.neuro.28.061604.135751

Denève, S., and Machens, C. K. (2016). Efficient codes and balanced networks. *Nat. Neurosci.* 19, 375–382. doi: 10.1038/nn.4243

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

Favorov, O. V, and Kelly, D. G. (1996). Stimulus-response diversity in local neuronal populations of the cerebral cortex. *Neuroreport* 7, 2293–2301. doi: 10.1097/00001756-199610020-00006

Földiák, P. (1990). Forming sparse representations by local anti-Hebbian learning. *Biol. Cybern.* 64, 165–170. doi: 10.1007/BF02331346

Földiák, P. (2002). "Sparse coding in the primate cortex," in *The Handbook of Brain Theory and Neural Networks*, *2nd Edn.*, ed M. A. Arbib (Cambridge, MA: MIT Press), 1064–1068.

Friauf, E., McConnell, S. K., and Shatz, C. J. (1990). Functional synaptic circuits in the subplate during fetal and early postnatal development of cat visual cortex. *J. Neurosci.* 10, 2601–2613.

Gemmeke, J. F., Virtanen, T., and Hurmalainen, A. (2011). Exemplar-based sparse representations for noise robust automatic speech recognition. *IEEE Trans. Audio Speech Lang. Processing* 19, 2067–2080. doi: 10.1109/TASL.2011.2112350

Ghosh, A., and Shatz, C. J. (1992). Involvement of subplate neurons in the formation of ocular dominance columns. *Science* 255, 1441–1443. doi: 10.1126/science.1542795

Gibson, J. R., Beierlein, M., and Connors, B. W. (1999). Two networks of electrically coupled inhibitory neurons in neocortex. *Nature* 402, 75–79. doi: 10.1038/47035

Gilbert, C. D., and Wiesel, T. N. (1992). Receptive field dynamics in adult primary visual cortex. *Nature* 356, 150–152. doi: 10.1038/356150a0

Gui, J., Sun, Z., Ji, S., Tao, D., and Tan, T. (2016). Feature selection based on structured sparsity: a comprehensive study. *IEEE Trans. neural networks Learn. Syst.* 28, 1490–1507. doi: 10.1109/TNNLS.2016.2551724

Habenschuss, S., Puhr, H., and Maass, W. (2013). Emergence of optimal decoding of population codes through STDP. *Neural Comput.* 25, 1371–1407. doi: 10.1162/NECO_a_00446

Hansel, D., and Mato, G. (2013). Short-term plasticity explains irregular persistent activity in working memory tasks. *J. Neurosci.* 33, 133–149. doi: 10.1523/JNEUROSCI.3455-12.2013

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

Hawkins, J., Ahmad, S., and Dubinsky, D. (2011). Cortical learning algorithm and hierarchical temporal memory. *Numenta Whitepaper*, 1–68. Available online at: http://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf

Hebb, D. (1949). *The Organization of Behavior: A Neuropsychological Theory*. New York, NY: John Wiley & Sons, Inc.

Hecht-Nielsen, R. (1990). *Neurocomputing*. Boston, MA: Addison-Wesley Pub. Co.

Horton, J. C., and Adams, D. L. (2005). The cortical column: a structure without a function. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 360, 837–862. doi: 10.1098/rstb.2005.1623

Hromádka, T., Deweese, M. R., and Zador, A. M. (2008). Sparse representation of sounds in the unanesthetized auditory cortex. *PLoS Biol.* 6:e16. doi: 10.1371/journal.pbio.0060016

Hu, T., Pehlevan, C., and Chklovskii, D. B. (2014). "A Hebbian/Anti-Hebbian network for online sparse dictionary learning derived from symmetric matrix factorization," in *2014 48th Asilomar Conference on Signals, Systems and Computers* (Pacific Grove, CA: IEEE), 613–619. doi: 10.1109/ACSSC.2014.7094519

Ibrayev, T., James, A., Merkel, C., and Kudithipudi, D. (2016). "A design of HTM spatial pooler for face recognition using Memristor-CMOS hybrid circuits," in *IEEE International Symposium on Circuits and Systems* (Montréal, QC). doi: 10.1109/ISCAS.2016.7527475

Ito, M. (1989). Long-term depression. *Annu. Rev. Neurosci.* 12, 85–102. doi: 10.1146/annurev.ne.12.030189.000505

Jones, E. G. (2000). Microcolumns in the cerebral cortex. *Proc. Natl. Acad. Sci. U.S.A.* 97, 5019–5021. doi: 10.1073/pnas.97.10.5019

Kaas, J. H. (1997). Topographic maps are fundamental to sensory processing. *Brain Res. Bull.* 44, 107–112. doi: 10.1016/S0361-9230(97)00094-4

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge, MA: The MIT Press.

Kanold, P. O., Kara, P., Reid, R. C., and Shatz, C. J. (2003). Role of subplate neurons in functional maturation of visual cortical columns. *Science* 301, 521–525. doi: 10.1126/science.1084152

Kremkow, J., Perrinet, L. U., Monier, C., Alonso, J.-M., Aertsen, A., Frégnac, Y., et al. (2016). Push-pull receptive field organization and synaptic depression: mechanisms for reliably encoding naturalistic stimuli in V1. *Front. Neural Circuits* 10:37. doi: 10.3389/fncir.2016.00037

Lavin, A., and Ahmad, S. (2015). "Evaluating real-time anomaly detection algorithms - the Numenta Anomaly Benchmark," in *14th International Conference on Machine Learning and Applications* (Miami, FL: IEEE ICMLA). doi: 10.1109/ICMLA.2015.141

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 2278–2324. doi: 10.1109/5.726791

Lee, H., Battle, A., Raina, R., and Ng, A. Y. (2006). "Efficient sparse coding algorithms" in *Advances in Neural Information Processing Systems* (Vancouver, BC), 801–808.

Mairal, J., Bach, F., Ponce, J., and Sapiro, G. (2010). Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res.* 11, 19–60.

Majani, E., Erlanson, R., and Abu-Mostafa, Y. (1988). "On the K-winners-take-all network," in *Proceedings of the First International Conference on Neural Information Processing Systems* (Lake Tahoe, NV), 634–642.

Major, G., Larkum, M. E., and Schiller, J. (2013). Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.* 36, 1–24. doi: 10.1146/annurev-neuro-062111-150343

Makhzani, A., and Frey, B. (2015). "k-sparse autoencoders," in *Advances in Neural Information Processing Systems 28* (Montreal, QC), 2791–2799.

Markram, H., Toledo-Rodriguez, M., Wang, Y., Gupta, A., Silberberg, G., and Wu, C. (2004). Interneurons of the neocortical inhibitory system. *Nat. Rev. Neurosci.* 5, 793–807. doi: 10.1038/nrn1519

Masquelier, T. (2013). Neural variability, or lack thereof. *Front. Comput. Neurosci.* 7:7. doi: 10.3389/fncom.2013.00007

Mnatzaganian, J., Fokoué, E., and Kudithipudi, D. (2017). A mathematical formalization of hierarchical temporal memory's spatial pooler. *Front. Robot. AI* 3:81. doi: 10.3389/frobt.2016.00081

Mountcastle, V. B. (1997). The columnar organization of the neocortex. *Brain* 120, 701–722. doi: 10.1093/brain/120.4.701

Nudo, R. J. (2013). Recovery after brain injury: mechanisms and principles. *Front. Hum. Neurosci.* 7:887. doi: 10.3389/fnhum.2013.00887

Olshausen, B. A., and Field, D. J. (1996a). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* 381, 607–609. doi: 10.1038/381607a0

Olshausen, B. A., and Field, D. J. (1996b). Natural image statistics and efficient coding. *Netw. Comput. Neural Syst.* 7, 333–339. doi: 10.1088/0954-898X_7_2_014

Olshausen, B. A., and Field, D. J. (1997). Sparse coding with an overcomplete basis set: a strategy employed by V1? *Vision Res.* 37, 3311–3325. doi: 10.1016/S0042-6989(97)00169-7

Olshausen, B. A., and Field, D. J. (2004). Sparse coding of sensory inputs. *Curr. Opin. Neurobiol.* 14, 481–487. doi: 10.1016/j.conb.2004.07.007

Pati, N., Pradhan, A., Kanoje, L. K., and Das, T. K. (2015). An approach to image compression by using sparse approximation technique. *Proc. Comput. Sci.* 48, 769–775. doi: 10.1016/j.procs.2015.04.213

Perrinet, L. U. (2010). Role of homeostasis in learning sparse representations. *Neural Comput.* 22, 1812–1836. doi: 10.1162/neco.2010.05-08-795

Pietron, M., Wielgosz, M., and Wiatr, K. (2016). "Formal analysis of HTM spatial pooler performance under predefined operation conditions," in *International Joint Conference on Rough Sets* (Olsztyn: Springer International Publishing), 396–405. doi: 10.1007/978-3-319-47160-0_36

Porter, J. T., Johnson, C. K., and Agmon, A. (2001). Diverse types of interneurons generate thalamus-evoked feedforward inhibition in the mouse barrel cortex. *J. Neurosci.* 21, 2699–2710.

Purdy, S. (2016). Encoding data for HTM systems. *arXiv* arXiv:1602.05925.

Sivaram, G. S. V. S., Nemala, S. K., Elhilali, M., Tran, T. D., and Hermansky, H. (2010). "Sparse coding for speech recognition," in *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (Dallas, TX: IEEE), 4346–4349. doi: 10.1109/ICASSP.2010.5495649

Song, S., Miller, K. D., and Abbott, L. F. (2000). Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nat. Neurosci.* 3, 919–926. doi: 10.1038/78829

Spruston, N. (2008). Pyramidal neurons: dendritic structure and synaptic integration. *Nat. Rev. Neurosci.* 9, 206–221. doi: 10.1038/nrn2286

Swadlow, H. A. (2002). Thalamocortical control of feed-forward inhibition in awake somatosensory "barrel" cortex. *Philos. Trans. R. Soc. Lond. B Biol. Sci.* 357, 1717–1727. doi: 10.1098/rstb.2002.1156

Teyler, T. J., and DiScenna, P. (1987). Long-term potentiation. *Annu. Rev. Neurosci.* 10, 131–161. doi: 10.1146/annurev.ne.10.030187.001023

Thornton, J., and Srbic, A. (2013). Spatial pooling for greyscale images. *Int. J. Mach. Learn. Cybern.* 4, 207–216. doi: 10.1007/s13042-012-0087-7

Tolhurst, D. J., Movshon, J. A., and Dean, A. F. (1983). The statistical reliability of signals in single neurons in cat and monkey visual cortex. *Vis. Res.* 23, 775–785. doi: 10.1016/0042-6989(83)90200-6

Udin, S. B., and Fawcett, J. W. (1988). Formation of topographic maps. *Annu. Rev. Neurosci.* 11, 289–327. doi: 10.1146/annurev.ne.11.030188.001445

Vinje, W. E., and Gallant, J. L. (2000). Sparse coding and decorrelation in primary visual cortex during natural vision. *Science* 287, 1273–1276. doi: 10.1126/science.287.5456.1273

Weliky, M., Fiser, J., Hunt, R. H., and Wagner, D. N. (2003). Coding of natural scenes in primary visual cortex. *Neuron* 37, 703–718. doi: 10.1016/S0896-6273(03)00022-9

Willmore, B., and Tolhurst, D. J. (2001). Characterizing the sparseness of neural codes. *Network* 12, 255–270. doi: 10.1080/net.12.3.255.270

Willshaw, D. J., Buneman, O. P., and Longuet-Higgins, H. C. (1969). Non-holographic associative memory. *Nature* 222, 960–962. doi: 10.1038/222960a0

Wonders, C. P., and Anderson, S. A. (2006). The origin and specification of cortical interneurons. *Nat. Rev. Neurosci.* 7, 687–696. doi: 10.1038/nrn1954

Wright, J., Yang, A. Y., Ganesh, A., Sastry, S. S., and Ma, Y. (2009). Robust face recognition via sparse representation. *IEEE Trans. Pattern Anal. Mach. Intell.* 31, 210–227. doi: 10.1109/TPAMI.2008.79

Yang, J., Wright, J., Ma, Y., and Huang, T. S. (2010). Image super-resolution via sparse representation. *IEEE Trans. Image Process.* 19, 2861–2873. doi: 10.1109/TIP.2010.2050625

Zito, K., and Svoboda, K. (2002). Activity-dependent synaptogenesis in the adult Mammalian cortex. *Neuron* 35, 1015–1017. doi: 10.1016/S0896-6273(02)00903-0

Zylberberg, J., Murphy, J. T., and DeWeese, M. R. (2011). A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of V1 simple cell receptive fields. *PLoS Comput. Biol.* 7:e1002250. doi: 10.1371/journal.pcbi.1002250

Conflict of Interest Statement: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Numenta has some patents relevant to the work. Numenta has stated that use of its intellectual property, including all the ideas contained in this work, is free for non-commercial research purposes. In addition Numenta has released all pertinent source code as open source under the AGPL V3 license (which includes a patent peace provision).