A comparative study of HTM and other neural network models for online sequence learning with streaming data

Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins
Numenta, Inc
Redwood City, CA, United States

Abstract— **Online sequence learning from streaming data is one of the most challenging topics in machine learning. Neural network models represent promising candidates for sequence learning due to their ability to learn and recognize complex temporal patterns. In this paper, we present a comparative study of Hierarchical Temporal Memory (HTM), a neurally-inspired model, and other feedforward and recurrent artificial neural network models on both artificial and real-world sequence prediction algorithms. HTM and long-short term memory (LSTM) give the best prediction accuracy. HTM additionally demonstrates many other features that are desirable for real-world sequence learning, such as fast adaptation to changes in the data stream, robustness to sensor noise and fault tolerance. These features make HTM an ideal candidate for online sequence learning problems.**

Keywords—HTM, sequence learning, streaming data analysis, online learning, LSTM, ESN, ELM

I. INTRODUCTION

Countless machine learning tasks require sequence learning and recognition of complex temporal patterns. A number of neural network models have been proposed to model sequential data. Feedforward networks, such as time delay neural networks (TDNN), have been used to model sequential data by adding a set of delays to the input [1]. Recurrent neural networks can model sequence structure with recurrent lateral connections and can process the data sequentially one record at a time. Long short-term memory (LSTM) has the ability to selectively pass information across time and can model very long term dependencies using gating mechanisms [2]. Echo state networks (ESN) use a randomly connected recurrent network as a dynamics reservoir and model a sequence as trainable linear combination of these response signals [3].

These neural networks are powerful learning models that achieve state-of-the-art results on a wide range of sequence learning benchmark tasks. Real-world streaming data analysis, however, presents unique challenge for these models. Most neural network models are trained in batch mode, they require storing a large set of sequences and an offline batch training to minimize error. This approach is not suitable for dealing with the increasing amount of streaming data, which requires real-time online learning and fast adaptation to new patterns in the data.

In this paper we present a comparative study of HTM sequence memory with these artificial neural network models. HTM is a brain-inspired neural network model of sequence learning based on computational principles of the cortex [4], [5]. Using sophisticated model neurons with many recently discovered properties of pyramidal cells and active dendrites [6], HTM can model complex sequences using sparse distributed temporal codes. The network is trained in real-time using an online unsupervised Hebbian-style learning rule. The algorithms have been applied to many practical problems, including discrete and continuous sequence prediction, anomaly detection, and sequence recognition and classification.

We compare HTM sequence memory with three popular neural network models, including two recurrent networks with batch training (LSTM, ESN), and a feedforward network with sequential online learning mode, extreme learning machine (ELM) [7], [8]. We show that HTM sequence memory achieves comparable prediction accuracy to these other techniques on streaming data analysis tasks. In addition it exhibits a flexibility that is missing from batch machine learning algorithms. We demonstrate that HTM networks can learn complex sequences, rapidly adapt to changing statistics in the data, and naturally handle multiple predictions and branching sequences.

The paper is organized as follows. In section 2, we discuss the challenges of streaming data analysis. In section 3, we introduce the HTM sequence memory model. In sections 4 and 5, we apply HTM and other neural network models to discrete artificial sequence learning and continuous real world time series prediction respectively. Discussion and conclusions are given in section 6. Implementation details are given in section 7.

II. CHALLENGES OF STREAMING DATA ANALYSIS

With the increasing availability of streaming data, there is an increasing demand for online sequence learning algorithms. Here, a data stream is an ordered sequence of data records that can be read only once using limited computing and storage capabilities. In the field of data stream mining, the goal is to extract knowledge from continuous data streams such as computer network traffic, sensor data, financial transactions, etc. [9]–[11], which often have changing statistics (non-stationary) [12]. For most machine learning benchmarks, the quality of a learning algorithm is typically evaluated as the prediction accuracy on a test dataset for a particular problem. This is often the

978-1-5090-0620-5/16/$31.00 ©2016 IEEE

sole judging criterion for time series prediction competitions [13], [14]. However, real-world sequence learning from complex, noisy data streams requires many other properties in addition to prediction accuracy. Below we list unique challenges for online sequence learning. A good sequence learning algorithm should not only have good prediction accuracy but also exhibit properties to solve these challenges.

A. Continuous online learning

Continuous data streams often have changing statistics. As a result, the algorithm needs to continuously learn and adapt to changes in the data. This property is important for processing continuous real-time sensory streams, but it is not incorporated in most machine learning algorithms. The vast majority of algorithms learn in batches. That is, these algorithms either store the entire set of sequences and analyze them offline or keep a large enough chunk of the sequence in a memory buffer. The batch-training paradigm not only requires more computing and storage sources, but also prevents an algorithm from rapidly adapting to changes in the data streams.

B. High-order predictions

Real-world sequences contain contextual dependencies that span multiple time steps. The algorithm needs to dynamically determine how much temporal context is required to make good predictions. We call this the ability to make high-order predictions, where the term "order" refers to Markov order, specifically the number of previous time steps the algorithm needs to consider in order to make accurate predictions. It is important to note that an ideal algorithm should learn the order automatically and efficiently.

C. Multiple simultaneous predictions

For a given temporal context, there could be multiple possible future outcomes. A good sequence learning algorithm should be able to make multiple predictions simultaneously and evaluate the likelihood of each prediction online. This requires the algorithm to output a distribution of possible future outcomes. This property is present in HMMs [15] and generative recurrent neural network models [2], but not in other approaches like ARIMA, which are limited to maximum likelihood prediction.

D. Noise robustness and fault tolerance

Real world sequence learning deals with noisy data sources where sensor noise, data transmission errors and inherent device limitations frequently result in inaccurate or missing data. A good sequence learning algorithm should exhibit robustness to noise in the inputs. The algorithm should also be able to learn properly in the event of system faults, such as loss of synapses and neurons in a neural network. Noise robustness and fault tolerance ensures flexibility and wide applicability of the algorithm to a wide variety of problems.

E. Requires little or no hyperparameter tuning

Most machine-learning algorithms require optimizing a set of hyperparameters to get good performance. This is known as the hyperparameter optimization or model selection problem. It typically involves searching through a manually specified subset of the hyperparameter space, guided by performance metrics on a cross-validation dataset. Hyperparameter tuning presents a major challenge for data stream mining, because the optimal set of hyperparameters may change as the data changes. The ideal sequence learning algorithm should have good performance on a wide range of problems without extensive task-specific hyperparameter tuning.

Although some algorithms meet one or two of the above criteria, a truly flexible and powerful system would meet all of them. In the rest of the paper, we will compare HTM sequence memory with three other popular neural network models that have been used for sequence learning, ELM, ESN and LSTM, on various sequence learning problems using the above criteria.

III. HTM SEQUENCE MEMORY

In this section we describe the computational details of HTM sequence memory. We first describe the single neuron model. We then describe the representation of high order sequences, followed by a formal description of our learning rules. A detailed mapping to the biology can be found in [4].

A. HTM neuron model

The HTM neuron (Fig. 1A) implements simple non-linear synaptic integration inspired by recent neuroscience findings regarding the function of cortical neurons and dendrites [6], [16]. Each neuron in the network contains two separate zones: a proximal zone containing a single dendritic segment and a distal zone containing a set of independent dendritic segments. Each segment maintains a set of synapses. The source of the synapses is different depending on the zone (Fig. 1A). Proximal synapses represent feed-forward inputs into the layer whereas distal synapses represent lateral connections within a layer.

Each distal dendritic segment contains a set of lateral synaptic connections from other neurons within the layer. A segment becomes active if the number of simultaneously active connections exceeds a threshold. An active segment does not cause the cell to fire but instead causes the cell to enter a depolarized state, which we call the "predictive state". In this way each segment detects a particular temporal context and makes predictions based on that context. Each neuron can be in one of three internal states: an active state, a predictive state, or a non-active state. The output of the neuron is always binary: it is either active or not.

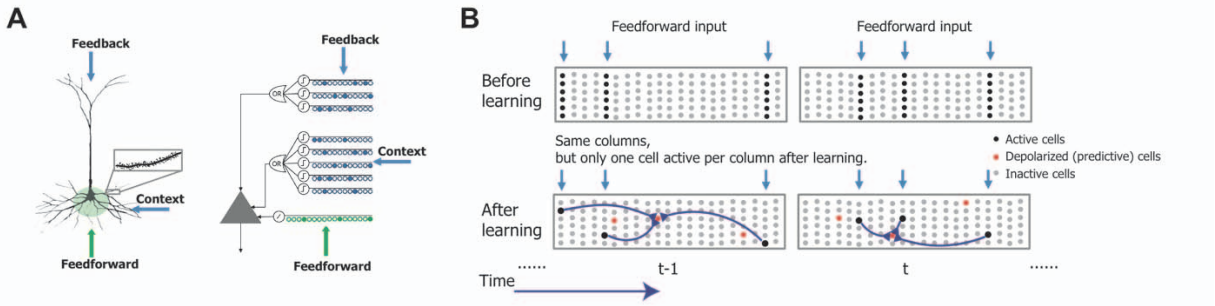

Figure 1 The HTM sequence memory model. **A**. An HTM neuron (*right*) has three distinct dendritic integration zones, corresponding to different parts of the dendritic tree of pyramidal neurons (*left*). An HTM neuron models dendrites as an array of coincident detectors, each with a set of synapses. The co-activation of a set of synapses on a distal dendrite will cause a dendritic spike and depolarize the soma (predicted state). **B.** States of the HTM network before (*top*) and after (*bottom*) learning.

B. Two separate sparse representations

The HTM network consists of a layer of HTM neurons organized into a set of columns (Fig. 1A). The network represents high-order sequences using a composition of two separate sparse representations. The first representation is at the column level. All neurons within a column detect identical feed-forward input patterns. Each input element is encoded as a sparse distributed activation of columns at any point in time (Fig. 1B, blue arrows). This sparse set of columns is chosen according to a spatial competitive learning rule [17].

The second representation is at the level of individual cells within these columns. After sequence learning (Fig. 1B, bottom), at any given time a distinct subset of cells in the active columns will represent information regarding the learned temporal context of the current pattern. These cells in turn lead to predictions of the upcoming input through lateral projections to other cells within the same network. The predictive state of a cell controls inhibition within a column. If a cell is in the predicted state and later receives sufficient feed-forward input, it will become active faster and inhibit other cells within that column. If there were no cells in the predicted state for an active column, all cells within the column become active.

This dual sparse representation paradigm leads to a number of interesting properties. First, because information is stored by co-activation of multiple cells in a distributed manner, the model is naturally robust to both noise in the input and system faults such as loss of neurons and synapses. Second, the use of sparse representations allows the model to make multiple predictions simultaneously. For example, if we present an input to the network without any context, all cells in columns representing that input will fire, which then leads to a prediction of multiple elements that could follow the current input elements. A detailed discussion on this topic can be found in [4].

C. HTM activation and learning rules

The previous sections provided an intuitive description of network behavior. In this section we describe the formal activation and learning rules for the HTM network. Consider a network with N columns and M neurons per column; we denote the activation state at time step t with an $M\times N$ binary matrix $\mathbf{A}^t$, where a_{ij}^t is the activation state of the i'th cell in the j'th column. Similarly, an $M\times N$ 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. We model learning in the HTM network with synapse growth [18], rather than synaptic weight adjustment. Each synapse is associated with a scalar permanence value, which corresponds to the proximity between the axonal terminal and the dendrite of the postsynaptic neuron. We consider a synapse connected if its permanence value exceeds a connection threshold. We use an $M\times N$ matrix $\mathbf{D}_{ij}^d$ to denote the permanence of d'th segment of the i'th cell in the j'th column and use a binary matrix $\tilde{\mathbf{D}}_{ij}^d$ to denote only the connected synapses. 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 randomly chosen such that about 50% of the potential synapses are connected initially.

The predictive state of the neuron is handled as follows: if a dendritic segment receives enough input, it becomes active and subsequently depolarizes the cell body without causing an immediate spike. Mathematically, the predictive state at time step t is calculated as follows:

$$\pi_{ij}^t = \begin{cases} 1 \text{ if } \exists_d \left\| \tilde{\mathbf{D}}_{ij}^d \circ \mathbf{A}^t \right\|_1 > \theta \\ 0 \text{ otherwise} \end{cases} \quad (1)$$

Threshold θ represents the segment activation threshold and $\circ$ represents element-wise multiplication. Since the distal synapses receive inputs from previously active cells in the same layer, it contains contextual information about future inputs (Fig. 1B).

At any time, an inter-columnar inhibitory process select a sparse set of columns that best match the current feed-forward input pattern. We calculate the number of active proximal synapses for each column, and activate the top 2% of the columns that receive the most synaptic inputs. We denote this set as $\mathbf{W}^t$. Neurons in the predictive state (i.e. depolarized) will have competitive advantage over other neurons receiving the same feed-forward inputs. Specifically, a depolarized cell fires faster than other non-depolarized cells if it subsequently receives sufficient feed-forward input. By firing faster, it prevents neighboring cells

in the same column from activating with intra-column inhibition. The active state for each cell is calculated as follows:

$$a_{ij}^t = \begin{cases} 1 \text{ if } j \in \mathbf{W}^t \text{ and } \pi_{ij}^{t-1} = 1 \\ 1 \text{ if } j \in \mathbf{W}^t \text{ and } \sum_i \pi_{ij}^{t-1} = 0 \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

The first conditional expression of Eq. 2 represents a cell in a winning column becoming active if it was in a predictive state during the preceding time step. If none of the cells in a winning column are in a predictive state, all cells in that column become active, as in the second conditional of Eq. 2.

The lateral connections in the sequence memory model are learned using a Hebbian-like rule. Specifically, if a cell is depolarized and subsequently becomes active, we reinforce the dendritic segment that caused the depolarization. If no cell in an active column is predicted, we select the cell with the most activated segment and reinforce that segment. Reinforcement of a dendritic segment involves decreasing permanence values of inactive synapses by a small value p^- and increasing the permanence for active synapses by a larger value p^+:

$$\Delta \mathbf{D}_{ij}^d = p^+ \dot{\mathbf{D}}_{ij}^d \circ \mathbf{A}^{t-1} - p^- \dot{\mathbf{D}}_{ij}^d \circ (\mathbf{1} - \mathbf{A}^{t-1}) \quad (3)$$

We also apply a very small decay to active segments of cells that did not become active, mimicking the effect of long-term depression (forgetting) [19]:

$$\Delta \mathbf{D}_{ij}^d = p^{--} \dot{\mathbf{D}}_{ij}^d \text{ where } a_{ij}^t = 0 \text{ and } \left\| \tilde{\mathbf{D}}_{ij}^d \circ \mathbf{A}^{t-1} \right\|_1 > \theta \quad (4)$$

where $p^{--} \ll p^-$.

A complete set of parameters and further implementation details can be found in the appendix. Notably, we used the same set of parameters for all the different types of sequence learning tasks in this paper.

D. SDR encoder and classifier

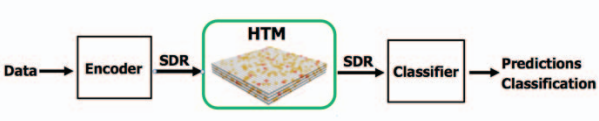

Figure 2 Functional steps for using HTM on real-world sequence learning tasks.

The HTM sequence memory operates with sparse distributed representations (SDRs) internally. To apply HTM to real-world sequence learning problems, we need to first convert the original data to SDRs using an encoder. We have created a variety of encoders to deal with different data types [20]. In this paper, we used a random SDR encoder for categorical data, and used scalar and date-time encoders for the taxi passenger prediction experiment.

To decode prediction values from the output SDRs of HTM, we considered two classifiers. For the single-step discrete sequence prediction task, we computed the overlap of the predicted cells with the SDRs of all observed elements and selected the one with the highest overlap. For the continuous scalar value prediction task, we used a single layer feedforward classification network with softmax activation function [21]. We trained the classifier online using a maximum likelihood learning rule.

IV. High-order sequence prediction

We conducted experiments to compare HTM sequence memory, LSTM, and online sequential extreme learning machine (OS-ELM) on high-order sequence learning tasks. The tasks evaluate the ability to perform online sequence learning, to adapt quickly to changes of the sequences, and to make multiple predictions simultaneously.

A. Online learning with streaming data

We created a discrete high-order temporal sequence dataset, where sequences are designed such that any learning algorithm would have to maintain context of at least the first two elements of each sequence in order to correctly predict the last element of the sequence (Fig. 3A). We encoded each symbol in the sequence as an SDR for HTM sequence memory, with 40 randomly chosen active bits in a vector of 2048 bits. This high dimensional representation does not work well for LSTM or ELM due to the large number of parameters required. Instead, we used a real-valued dense distributed representation for LSTM [2]. Each symbol is encoded as a 25-dimensional vector with each dimension's value randomly chosen from [-1, 1].

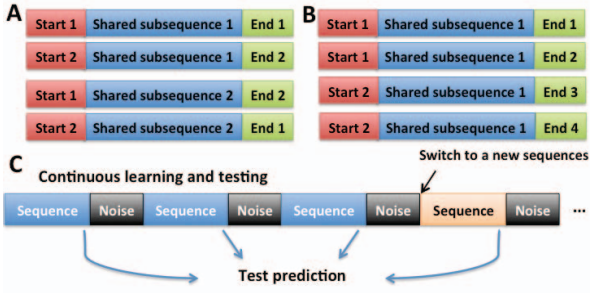

Figure 3 Design of the high-order sequence prediction task. **A.** Structure of high order sequences with shared subsequences. **B.** High order sequences with multiple possible endings. **C.** Stream of sequences with noise between sequences. Both learning and testing occur continuously. After the model learned one set of sequences, we switched to a new set of sequences with contradictory endings to test the adaptation to changes in the data stream.

We used the sequence dataset in a continuous streaming scenario (Fig. 3C). At the beginning of a trial, we randomly chose a sequence from the dataset and sequentially presented each of its elements. At the end of each sequence, we presented a single random element (not among the set of sequences) to the model, representing random noise between sequences. This is a difficult learning problem, since sequences are embedded in random noise; the start and end points are not marked. We tested the algorithms for predicting the last element of each sequence continuously as the algorithm observed a stream of sequences, and reported the percentage of correct predictions over time.

Because the sequences are presented in a streaming fashion and predictions are required continuously, this task represents a continuous online learning problem. The HTM sequence memory is naturally suitable for online learning as

it learns from every new data point and does not require the data stream to be broken up into predefined chunks. ELM also has a well-established online sequential learning model [7]. LSTM, however, typically requires batch training to reliably calculate the error signal. Thus, to test LSTM in this online learning task, LSTM networks were retrained at regular intervals on a buffered dataset of the previous time steps. The experiments include several LSTM models with varying learning window (buffer) sizes.

To quantify model performance, we classified the state of the model before presenting the last element of each sequence to retrieve the top K predictions, where $K = 1$ for the single prediction case and $K = 2$ or 4 for the multiple predictions case (Fig. 3B). We considered the prediction correct if the actual last element was among the top K predictions of the model. Since these are online learning tasks, there are no separate training and test phases. Instead, we continuously report the prediction accuracy of the end of each sequence before the model has seen it.

In the single prediction experiment (Fig. 4, left of the black dashed line), each sequence in the dataset has only one possible ending given its high-order context. The HTM sequence memory quickly achieves perfect prediction accuracy on this task (Fig. 4, red). Given a large enough learning window, LSTM also learns to accurately predict the high-order sequences (Fig. 4, green). Despite comparable model performance, HTM and LSTM use the data in different ways: LSTM requires many passes over the learning window and is retrained to perform gradient-descent optimization each time, whereas HTM only needs to see each element once. As a fair comparison, ELM trained in an online, sequential fashion and performed one-pass learning. However it learned the sequences slower than HTM, and never achieved perfect performance (Fig. 4, blue). HTM is the only algorithm that achieved consistently perfect prediction accuracy on this task.

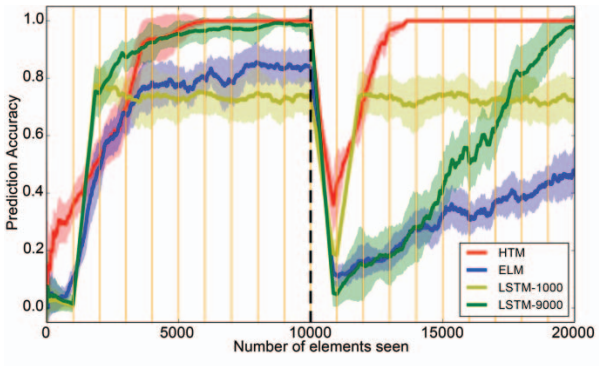

Figure 4 Prediction accuracy of HTM (red), ELM (blue) and LSTM (yellow, green) on an artificial dataset. Prediction accuracy is calculated as a moving average over the last 100 elements. HTM and ELM see each element once, and learn continuously; LSTM is retrained every 1000 elements (vertical orange lines), either on the last 1000 elements (yellow) or 9000 elements (green). Shaded area represents standard deviation of accuracy, calculated by using different sets of sequences with the same structure. The sequences are changed after 10,000 elements have been seen (black dashed line).

B. Adaptation to changes in the data stream

Once the models have achieved stable performance, we altered the dataset by swapping the last elements of pairs of high-order sequences (Fig. 2, black dashed line). The new sequences are thus contradictory to the old sequences. This forces the model to forget the old sequences and subsequently learn the new ones. The HTM sequence memory quickly recovers from the modification. In contrast, it takes a long time for LSTM and ELM to recover from the modification. Although using a smaller learning window can speed up the recovery (Fig. 4, *yellow*), it also causes worse prediction performance due to limited number of training samples.

A summary of the model performance on the high-order sequence prediction task is shown in Fig. 5. In general, there is a tradeoff to be made with LSTM or any other algorithm that requires batch training: a shorter learning window is required to adapt to changes of the data faster, but a longer learning window is required to perfectly learn high-order sequences (Fig. 5, green vs. yellow). The ELM has worse performance than HTM and LSTM (with long buffer size), and requires more data to achive the final accuracy (Fig. 5, blue). Although ELM does not require a buffered dataset for training and runs much faster than both LSTM and HTM, it does require the user to specify the maximal lag, which limits the maximum sequence order it can learn. The HTM sequence memory model dynamically learns high-order sequences without requiring a learning window or a maximum sequence length. It achieved the best final prediction accuracy with a small number of data samples. HTM's recovery is much faster than ELM and LSTM after the modification to the sequences, demonstrating its ability to adapt quickly to changes in data streams.

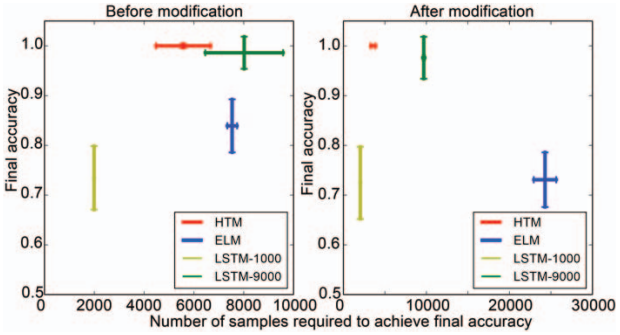

Figure 5 Final prediction accuracy as a function of the number of samples required to achieve final accuracy before (left) and after (right) modification of the sequences. Error bars represent standard deviation.

C. Simultaneous multiple predictions

In the experiment with multiple predictions, each sequence in the dataset has 2 or 4 possible endings, given its high-order context. The HTM sequence memory model rapidly achieves perfect prediction accuracy for both the 2-predictions and the 4-predictions cases. While only these two cases are shown, in reality HTM is able to make many multiple predictions correctly if the dataset requires it.

Given a large learning window, LSTM is able to achieve good prediction accuracy for the 2-predictions case, but when the number of predictions is increased to 4 or greater, it is not able to make accurate predictions.

HTM sequence memory is able to simultaneously make multiple predictions due to its use of SDRs. Because there is little overlap between two random SDRs, it is possible to predict a union of many SDRs and classify a particular SDR as being a member of the union with low chance of false positive error [22]. On the other hand, the real-valued dense distributed encoding used in LSTM and ELM is unsuitable for multiple predictions, because the average of multiple dense representations in the encoding space is not necessarily close to any of the component encodings, especially when the number of predictions being made is large. The problem can be solved by using local one-hot representations to code target inputs, but such representations have very limited capacity and do not work well when the number of possible inputs is large or unknown upfront.

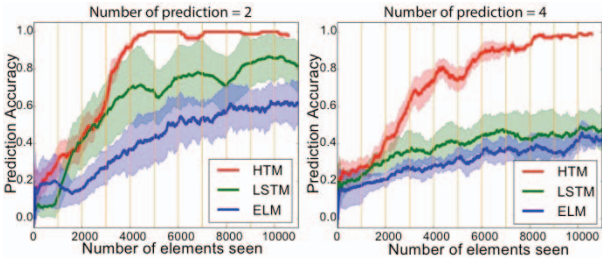

Figure 6 Performance on high order sequence prediction tasks that require two (left) or four (right) simultaneous predictions. Shaded regions represent standard deviations (calculated with different sets of sequences).

D. Fault tolerance

We tested the robustness of ELM, LSTM and HTM network with respective to removal of neurons. This fault tolerance property is important for hardware implementations of neural network models. After both networks achieved perfect performance on the high-order sequence prediction task (at the black dashed line, Fig. 2), we eliminated a fraction of the cells and their associated synaptic connections from the network. We then measured the prediction accuracy of both networks on the same data streams for an additional 5000 steps without further learning. There is no impact on HTM sequence memory model performance at up to 30% cell death, whereas performance of ELM and LSTM network declined rapidly with small fraction of cell death (Fig. 3). Similar sensitivity on system faults have also been observed on feedforward neural networks [23]. Although the fault tolerance of LSTM or other artificial neural network may be improved by introducing redundancy (replicating trained network) [24] or by special training method [25], the fault tolerance of HTM is naturally derived from properties of sparse distributed representations [22], in analogy to biological neural networks.

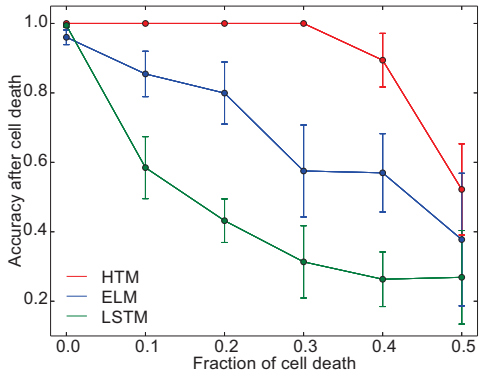

Figure 7 Robustness of the network to damage. The prediction accuracy after cell death is shown as a function of the fraction of cells that were removed from the network.

V. Prediction of Taxi Passenger Count

In order to compare the performance of HTM sequence memory with other sequence learning techniques in real-world scenarios, we considered the problem of predicting taxi passenger demand. Specifically, we aggregated the passenger counts in New York City taxi rides at 30-minute intervals using a public data stream provided by the New York City Transportation Authority [1]. This leads to sequences exhibiting rich patterns at different time scales (Fig. 8). The task is to predict the taxi passenger demand five steps (2.5 hours) in advance. This problem is an example of a large class of sequence learning problems that require rapid processing of streaming data to deliver information for real-time decision making [9], [10].

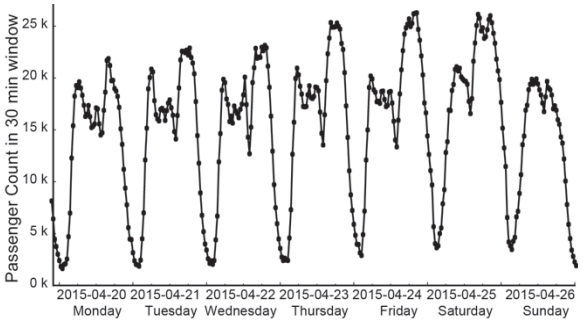

Figure 8 Prediction of the New York City taxi passenger data. *left*. Example portion of taxi passenger data (aggregated at 30 min intervals).

We applied HTM sequence memory and other neural network models to this problem. As in the previous experiment, we converted the original LSTM batch-training algorithm to an online learning algorithm by re-training the LSTM network on every week of data with a buffered dataset of the previous 1000, 3000, or 6000 samples. The ESN, ELM and HTM models were trained online. We used a time lag of 100 steps for the ELM model. The parameters of the ESN and LSTM network were extensively hand-tuned to provide the best possible accuracy on this dataset. The HTM model did not undergo any parameter tuning – it

[1] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

uses the same parameters that were used for the previous artificial sequence task.

We evaluated model performance using mean absolute percentage error (MAPE), an error metric robust to outliers in the prediction. We found that the HTM sequence memory had comparable performance to LSTM on the MAPE metrics. Both techniques had lower error than ELM and ESN. Note that HTM sequence memory achieved this performance with a single-pass training paradigm, whereas LSTM required multiple-passes on a buffered dataset.

Table 1 Average prediction error—mean absolute percentage error (MAPE)—on taxi passenger count data after observing 10,000 data records.

Model name	MAPE (%)
ELM (max lag 100)	15.7%
ESN	9.6%
LSTM (batch size 1000)	9.9%
LSTM (batch size 3000)	8.8%
LSTM (batch size 6000)	8.2%
HTM	7.8%

We then tested how fast different sequence learning algorithms can adapt to changes in the data. We artificially modified the data by decreasing weekday morning traffic (7am-11am) by 20% and increasing weekday night traffic (9pm-11pm) by 20% starting from April 1st. These changes in the data caused an immediate increase in prediction error for both HTM and LSTM (Fig. 9). Nevertheless, the prediction error of HTM sequence memory quickly recovered quickly, whereas the LSTM prediction error stayed high for much longer. As a result, HTM sequence memory had better prediction accuracy than LSTM on all error metrics after the data modification (Fig. 9).

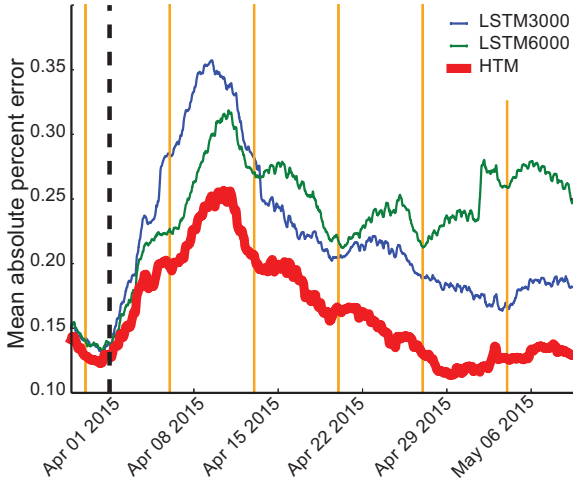

Figure 9 Prediction accuracy of LSTM and HTM after introduction of new patterns. A. The mean absolute percent error of HTM sequence memory (*red*) and LSTM networks (*green*, *blue*) after artificial manipulation of the data (*black dashed line*). The LSTM networks are re-trained every week at the orange vertical lines.

VI. Discussion and Conclusions

In this paper we presented a comparative study of HTM sequence memory, a novel neural network model for real-time sequence learning on time-varying input streams. Online learning from noisy data streams with continuously changing statistics presents unique challenges for sequence learning models. Using a variety of artificially generated and real-world sequence learning tasks, we presented and discussed prediction accuracy as well as other features of different neural network models that affect performance on real-world streaming applications. These properties govern the overall flexibility of an algorithm and are missing from previous sequence learning algorithms. HTM sequence memory satisfies these properties, and solves practical sequence modeling problems.

A. Continuous learning with streaming data

Most supervised sequence learning algorithms use a batch-training paradigm, where a cost function, such as prediction error, is minimized on a batch of training dataset. The LSTM model we compared in this paper, belong to this category. Although we can train these algorithms continuously using a sliding window [26], the batch-training paradigm is not a good match for time-series prediction on continuous streaming data. A small window may not contain enough training samples for learning complex sequences, while a large window introduces a limit on how fast the algorithm can adapt to changing statistics in the data. It is possible to use a smooth forgetting mechanism instead of hard retraining [27], [28], but this requires the user to tune parameters governing the forgetting speed to achieve good performance.

HTM sequence memory adopts a continuous learning paradigm. The model does not need to store a batch of data as the "training dataset". Instead, it learns from each data point using unsupervised Hebbian-like associative learning mechanisms. As a result the model rapidly adapts to changing statistics in the data. Although ELM also has an online sequential training algorithm [7], it is derived from the batch training algorithm and does not share the flexibility and fast adaptation properties of HTM.

We focused on comparing neural network models in this study. Other types of models and algorithms have also been developed in the past to solve streaming data mining. Evolving intelligent systems are capable of learning from changing environment by assuming online adaptation of both system structure and parameters [29], [30]. It would be interesting for future studies to compare HTM and other neural network models with evolving systems.

B. The use of sparse distributed representaitons

A key difference between HTM sequence memory and other neural network models is the use of sparse distributed representations (SDRs). In the cortex, information is primarily represented by the strong activation of a small set of neurons at any time, known as sparse coding [31]. HTM sequence memory uses SDRs to represent temporal sequences. Based on mathematical properties of SDRs [22], [32], each neuron in the HTM sequence memory model can

robustly learn and classify a large number of patterns under noisy conditions [4]. A rich distributed neural representation for temporal sequences emerges from computation in HTM sequence memory. Although we focus on sequence prediction in this paper, this representation is valuable for a number of tasks, such as anomaly detection and sequence classification [33].

C. Robustness and generalizations

An intelligent learning algorithm should be able to automatically handle a large variety of problems, yet most machine learning algorithms require a task-specific parameter search when applied to a novel problem. Learning in the cortex does not require an external tuning mechanism, and the same cortical region can be used for different functional purposes if the sensory input changes [34], [35]. Using computational principles derived from the cortex, we show that HTM sequence memory achieves performance comparable to LSTM networks on very different problems using the same set of parameters. These parameters were chosen according to known properties of real cortical neurons [4] and basic properties of sparse distributed representations [22]. Ideally, systems should be robust with respect to parameter sensitivity and easily applied to a wide range of problems.

VII. Implementation details

In our software implementation, we made a few simplifying assumptions to speed up simulation for large networks. We did not explicitly initialize a complete set of synapses across every segment and every cell. Instead, we greedily created segments on the least used cells in an unpredicted column and initialized potential synapses on that segment by sampling from previously active cells. This happened only when there is no match to any existing segment.

Because the HTM sequence model operates with sparse distributed representations (SDRs), specialized encoders are required to encode real-world data into SDRs. For the artificial datasets with categorical elements, we simply encoded each symbol in the sequence as a random SDR, with 40 randomly chosen active bits in a vector of 2048 bits.

For the NYC taxi dataset, three pieces of information were fed into the HTM model: raw passenger count, the time of day, and the day of week (ELM, ESN, and LSTM received the same information as input). We used NuPIC's standard scalar encoder to convert each piece of information into an SDR [20]. The encoder converts a scalar value into a large binary vector with a small number of ON bits clustered within a sliding window, where the center position of the window corresponds to the data value. This encoding was done for each of three data fields, and we subsequently combined the three SDRs via a competitive sparse spatial pooling process, resulting in 40 active bits in a vector of 2048 bits—as in the artificial dataset. The spatial pooling process is described in detail here [17].

The HTM sequence memory model used an identical set of model parameters for all the experiments described in the paper. A complete list of model parameters is shown below. The full source code for the implementation is available on Github at https://github.com/numenta/nupic

Table 2 HTM model parameters

Parameter name	Value
Number of columns N	2048
Number of cells per column M	32
Dendritic segment activation threshold θ	15
Initial synaptic permanence	0.21
Connection threshold for synaptic permanence	0.5
Synaptic permanence increment p^+	0.1
Synaptic permanence decrement p^-	0.1
Synaptic permanence decrement for predicted inactive segments p^{--}	0.01

We used the online sequential learning algorithm for ELM [7]. The network used 50 hidden neurons and a time lag of 100 for the taxi data and 200 hidden neurons and a time lag of 10 for the artificial dataset. We used 100 internal units, a spectral radius of 0.1, a teacher scaling of 0.01 and a learning rate of 0.1 for the ESN model. The parameters were hand tuned to achieve the best performance. The LSTM network had 20 fully connected cells were trained every 1000 iterations in batch mode until convergence.

We used mean absolute percentage error (MAPE) metric to evaluate the prediction accuracies of the models, an error metric that is less sensitive to outliers than root mean squared error;

$$\text{MAPE} = \frac{\sum_{t=1}^{N} |y_t - \hat{y}_t|}{\sum_{t=1}^{N} |y_t|} \tag{5}$$

where y_t is the observed data at time t, $\hat{y}_t$ is the model prediction for the data observed at t, and N is dataset length.

VIII. Reference

[1] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust.*, vol. 37, no. 3, pp. 328–339, Mar. 1989.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[3] H. Jaeger and H. Haas, "Harnessing nonlinearity: predicting chaotic systems and saving energy in wireless communication.," *Science*, vol. 304, no. 5667, pp. 78–80, Apr. 2004.

[4] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Front. Neural Circuits*, vol. 10, Mar. 2016.

[5] Y. Cui, C. Surpur, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," p. arXiv:1512.05463 [cs.NE], 2015.

[6] G. Major, M. E. Larkum, and J. Schiller, "Active properties of neocortical pyramidal neuron dendrites.," *Annu. Rev. Neurosci.*, vol. 36, pp. 1–24, 2013.

[7] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks.," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–23, Nov. 2006.

[8] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[9] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams," *ACM SIGMOD Rec.*, vol. 34, no. 2, p. 18, Jun. 2005.

[10] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, 2000, pp. 71–80.

[11] J. Gama, *Knowledge discovery from data streams*. Boca Raton, Florida: Chapman and Hall/CRC, 2010.

[12] M. Sayed-Mouchaweh and E. Lughofer, *Learning in non-stationary environments: methods and applications*. New York: Springer, 2012.

[13] S. Ben Taieb, G. Bontempi, A. F. Atiya, and A. Sorjamaa, "A review and comparison of strategies for multi-step ahead time series forecasting based on the NN5 forecasting competition," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7067–7083, Jun. 2012.

[14] S. F. Crone, M. Hibon, and K. Nikolopoulos, "Advances in forecasting with neural networks? Empirical evidence from the NN3 competition on time series prediction," *Int. J. Forecast.*, vol. 27, no. 3, pp. 635–660, Jul. 2011.

[15] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, Jan. 1986.

[16] N. Spruston, "Pyramidal neurons: dendritic structure and synaptic integration.," *Nat. Rev. Neurosci.*, vol. 9, no. 3, pp. 206–21, Mar. 2008.

[17] J. Hawkins, S. Ahmad, and D. Dubinsky, "Cortical learning algorithm and hierarchical temporal memory," *Numenta Whitepaper*, 2011. [Online]. Available: http://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf.

[18] D. B. Chklovskii, B. W. Mel, and K. Svoboda, "Cortical rewiring and information storage.," *Nature*, vol. 431, no. 7010, pp. 782–8, Oct. 2004.

[19] P. V Massey and Z. I. Bashir, "Long-term depression: multiple forms and implications for brain function.," *Trends Neurosci.*, vol. 30, no. 4, pp. 176–84, Apr. 2007.

[20] S. Purdy, "Encoding Data for HTM Systems," *arXiv*, p. 1602.05925, 2016.

[21] J. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing − Algorithms, Architectures and Applications.*, Springer-Verlag, Berlin, 1989, pp. 227–236.

[22] S. Ahmad and J. Hawkins, "Properties of sparse distributed representations and their application to hierarchical temporal memory," *arXiv*, p. 1503.07469, Mar. 2015.

[23] V. Piuri, "Analysis of Fault Tolerance in Artificial Neural Networks," *J. Parallel Distrib. Comput.*, vol. 61, no. 1, pp. 18–48, Jan. 2001.

[24] E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, "Investigating the fault tolerance of neural networks.," *Neural Comput.*, vol. 17, no. 7, pp. 1646–64, Jul. 2005.

[25] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv*, p. 1207.0580, 2012.

[26] T. Sejnowski and C. Rosenberg, "Parallel networks that learn to pronounce English text," *J. Complex Syst.*, vol. 1, no. 1, pp. 145–168, 1987.

[27] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, Jun. 1989.

[28] E. Lughofer and P. Angelov, "Handling drifts and shifts in on-line data streams with evolving fuzzy systems," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2057–2068, Mar. 2011.

[29] P. Angelov and N. Kasabov, "Evolving computational intelligence systems," in *Proc 1st International Workshop on Genetic Fuzzy Systems*, 2005, pp. 76–82.

[30] E. Lughofer, *Evolving fuzzy systems --- methodologies, advanced concepts and applications*. Springer Berlin Heidelberg, 2011.

[31] B. A. Olshausen and D. J. Field, "Sparse coding of sensory inputs," *Current Opinion in Neurobiology*, vol. 14. pp. 481–487, 2004.

[32] P. Kanerva, *Sparse Distributed Memory*. The MIT Press, 1988.

[33] 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)*, 2015.

[34] N. Sadato, A. Pascual-Leone, J. Grafman, V. Ibañez, M. P. Deiber, G. Dold, and M. Hallett, "Activation of the primary visual cortex by Braille reading in blind subjects.," *Nature*, vol. 380, no. 6574, pp. 526–8, Apr. 1996.

[35] J. Sharma, A. Angelucci, and M. Sur, "Induction of visual orientation modules in auditory cortex.," *Nature*, vol. 404, no. 6780, pp. 841–7, Apr. 2000.