Hierarchical Temporal Memory based One-pass Learning for Real-Time Anomaly Detection and Simultaneous Data Prediction in Smart Grids

Anomadarshi Barua, *Student Member, IEEE,* Deepan Muthirayan, *Member, IEEE,*
Pramod P. Khargonekar, *Fellow, IEEE,* Mohammad Abdullah Al Faruque, *Senior Member, IEEE*

Abstract—A neuro-cognitive inspired architecture based on the Hierarchical Temporal Memory (HTM) is proposed for anomaly detection and simultaneous data prediction in real-time for smart grid μPMU data. The key technical idea is that the HTM learns a *sparse distributed temporal representation* of sequential data that turns out to be very useful for anomaly detection and simultaneous data prediction in real-time. Our results show that the proposed HTM can predict anomalies within 83% - 90% accuracy for three different application profiles, namely *Standard, Reward Few False Positive, Reward Few False Negative* for two different datasets. We show that the HTM is competitive to five state-of-the-art algorithms for anomaly detection. Moreover, for the multi-step prediction in the online setting, the same HTM achieves a low 0.0001 normalized mean square error, a low negative log-likelihood score of 1.5 and is also competitive to six state-of-the-art prediction algorithms. We demonstrate that the same HTM model can be used for *both the tasks* and can learn online in one-pass, in an unsupervised fashion and adapt to changing statistics. *The other state-of-the-art algorithms are either less accurate or are limited to one of the tasks or cannot learn online in one-pass, and adapt to changing statistics.*

Index Terms—smart grid, anomaly detection, simultaneous prediction, hierarchical temporal memory, sparse distributed representation

1 Introduction

Smart grids are cyber-physical systems (CPSs) that are comprised of pervasive sensing, computation, and control in spatially distributed power networks. Such smart grids generate large volumes of data in real-time. Thus, the challenge and the opportunity lies in systems and algorithms that can extract useful information from the various data streams and make reliable decisions in (near) real-time.

Micro Phasor Measurement Units (μPMUs) are deployed in distribution networks of smart grids to provide rich data on voltage and current variations at a finer resolution. The operators can monitor the distribution applications in real-time, due to the high performance of μPMU technology in distribution networks. They support a wide range of control and diagnostic applications, such as real-time anomaly detection and data prediction. In this work, we consider the specific problem of detecting anomalies and simultaneously predicting future observations in smart grids from unlabelled data that is generated by μPMUs. Anomaly detection is an important problem because failing to detect anomalies in a timely manner can affect the whole system and cause massive power failures, and prediction can help in planning and control.

The data provided by μPMUs have a few characteristics that are relevant to the problem at hand. **First**, it provides time-series data that can be observed only one at a time in the sequential order they arrive. Hence, the data are not naturally suitable for batch learning as a full dataset is not available. **Second**, the smart grid is inherently dynamic and the statistics of the generated data can change over time (*i.e., concept drift*). **Third**, the μPMU data has inherent information of the smart grid dynamics which can be leveraged to predict future observations. Hence, the problems of anomaly detection and prediction are challenging for the following reasons: (i) the algorithm should be able to learn online in *one-pass*, and in an unsupervised fashion (i.e., without human intervention); (ii) should be able to learn continuously, i.e., handle concept drifts in data, and (iii) the same algorithm should be able to perform anomaly detection and data prediction in real-time.

In this work, we introduce an architecture based on Hierarchical Temporal Memory (HTM) [1] to address the aforementioned challenges. *To the best of our knowledge, this is the first paper, which demonstrates that anomaly detection and data prediction in smart grids can be performed using the same algorithm with a competitive accuracy and simultaneously in real-time, while learning from just one-pass and in an unsupervised fashion.*

2 Related Works and Contributions

2.1 Related works

Anomaly detection and data prediction have been studied extensively but independently in the smart grid domain. Moghaddass et al. [2] proposed a framework to detect anomalies at the customer level based on smart meter data. Zhou et al. [3] demonstrated the efficacy of *Ensemble-based* algorithm for online and robust anomaly detection in PMU data. G. Napier et al. [4] used a model-based approach to detect anomalies in the SCADA network in the smart grid. *The main limitation of these approaches is that their applicability*

• *Anomadarshi Barua[1], Deepan Muthirayan[2], Pramod P. Khargonekar[3], Mohammad Abdullah Al Faruque[4] are with the Department of Electrical Engineering and Computer Science, University of California, Irvine. E-mail: anomadab@uci.edu[1], deepan.m@uci.edu[2], pramod.khargonekar@uci.edu[3], alfaruqu@uci.edu[4]*

is limited to stationary conditions and they can not perform simultaneous prediction.

Different data-driven techniques have also been proposed for analyzing μPMU data of the smart grid. Supervised learning methods, such as decision trees [5], SVMs [6] and unsupervised learning methods, such as clustering [7] have already been proposed. Valenzuela et al. [8] used *Principal Component Analysis* (PCA) to classify power flow into regular and irregular sub-spaces to detect intrusion. Zhou et al. [9] proposed a semi-supervised approach using *kernel Principal Component Analysis* (kPCA) and *partially-hidden-structured SVM* (pSVM) to detect abnormal events in smart grids. *The drawbacks of these approaches are* that they are suitable for batch learning and can not be used for other tasks such as prediction. Brahma et al. [10] propose a dynamic real-time framework named as *Shapelets* that can accurately and speedily classify PMU data. Auto-Regressive Moving Average (ARMA) is widely used for anomaly detection and short term prediction for load forecasting [11]. Gao et al. [12] presented a dynamic state prediction method based on the Auto-Regressive (AR) Model using PMU data. Sia et al. [13] used *the Hurst exponent* to anticipate future voltage collapse using PMU signals. *The limitation of these approaches is that these models can not handle concept drifts and can not predict future observations simultaneously.* Moreover, Yang et al. [14], [15] proposed a deep PDS-ERT based learning method to realize real-time anomaly detection; however, this method requires batch learning and cannot learn in one-pass and unsupervised fashion.

Hollingsworth et al. [16] investigated the combination of Long-Short-Term-Memory (LSTM) and Auto-Regressive Integrated Moving Average (ARIMA) to detect anomalies and simultaneously forecast energy consumption. But this method requires large datasets to train and it is not a real-time, one-pass, and unsupervised learning method. Ahmad et al. [17] demonstrated the efficacy of the HTM for real-time anomaly detection for different applications. The fundamental difference between our work and [17] is that our work demonstrates that the HTM model, which can detect real-time anomalies, can also be reused to simultaneously predict future observations. Finally, methods used by industries like Netflix's *Robust Principle Component Analysis* (RPCA) [18] and Yahoo's EGADS [19] also require batch training, and so are not applicable for the online setting that we consider. *Most importantly, none of these approaches can be used simultaneously for the prediction task.*

2.2 Contributions

Our novel technical contributions are as follows:

- We introduce an architecture for anomaly detection and simultaneous data prediction in smart grids that is inspired by neuro-cognitive mechanisms of the human called Hierarchical Temporal Memory (HTM). The HTM is a powerful framework developed by Numenta for sequence learning. At the heart of the HTM is a Cortical Learning Algorithm (CLA) that is inspired by how the human neocortex functions [1]. To the best of our knowledge, we are the first to introduce this method for applications on smart grid μPMU data.
- We demonstrate the effectiveness and applicability of the HTM approach for addressing the challenges of learning online from smart grid μPMU data for anomaly detection and simultaneous data prediction. It has been demonstrated that the HTM has the capability for continuous and unsupervised learning from streaming data and has been proven to work well for real-time detection in other domains [17]. This justifies extending this approach to the same problems in smart grids.
- To demonstrate the effectiveness of the proposed approach, we compare the performance of the HTM with five state-of-the-art real-time anomaly detection algorithms, such as Random Cut Forest, Bayesian Change Point, Windowed Gaussian, EXPoSE, and Relative Entropy on μPMU data of the smart grid. We have used the Numenta Anomaly Benchmark (NAB) [20] to compare these anomaly detection algorithms. To the best of our knowledge, this metric is the first of its kind for smart grid data.
- For the prediction problem, we compare the performance of the HTM with a total of six state-of-the-art sequence learning and prediction algorithms for data prediction, such as online LSTMs, online LSTMs with 6000 buffer points, online LSTMs with 3000 buffer points, online LSTMs with 1000 buffer points, Time Delayed Neural network (TDNN), and Adaptive Filter. This performance comparison among all these models on μPMU data is also the first of its kind in the smart grid domain, to the best of our knowledge[1].

The remainder of this paper is organized as follows. Section 3 introduces the HTM model and its learning algorithm with an illustrative example of sequence learning. Section 4 demonstrates the application of the HTM based learning model for anomaly detection in real-time in smart grid μPMU data. This section also explains how the HTM based method detects both temporal and spatial anomalies, learns in a continuous-online fashion and provides the comparison of the performance with five other state-of-the-art real-time anomaly detection algorithms. Section 5 demonstrates that the same HTM from the previous section can be reused for multi-step prediction and compares the performance with six other state-of-the-art sequence prediction algorithms. Finally, limitations and conclusions are drawn in Section 6 and Section 7, respectively.

3 HTM Architecture and Learning Algorithm

In this section, we briefly discuss the structure of an HTM model. We then discuss the activation and the learning algorithm and illustrate the temporal representations learnt by an HTM through an example.

3.1 Hierarchical Temporal Memory (HTM)

3.1.1 Human neocortex and HTM neuron

The human brain (Fig.1(a)) has primarily three parts: neocortex, limbic part, and reptilian part. The limbic part handles the emotional feeling, the reptilian part supports survival instincts and the neocortex is responsible for human learning, cognition, and perception. The pyramidal neuron

1. The source code of this paper is available in the following link: https://github.com/unknown-commits/HTM_upmudata_anomaly_detection_prediction

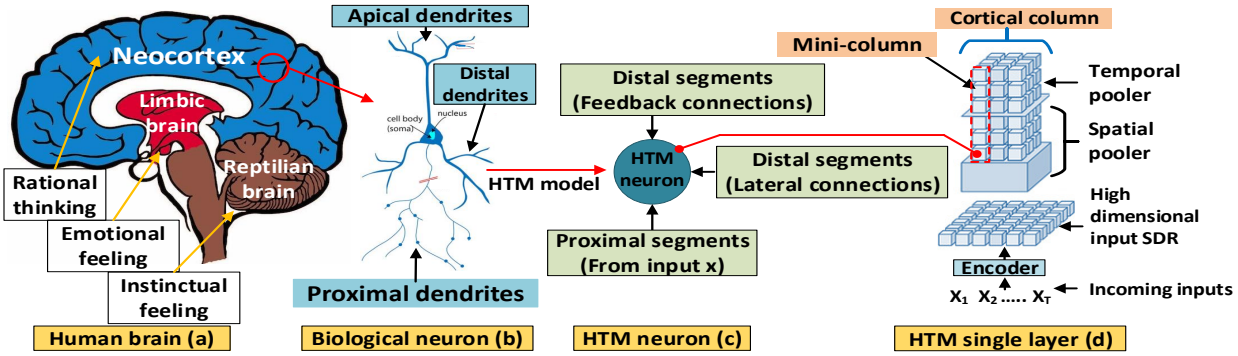

Fig. 1: Neocortical architecture of the Hierarchical Temporal Memory.

cell (Fig.1(b)) is the unit element of the neocortex and all the neurons in the neocortex are similar [21]. The functionality of this biological neuron is replicated in the HTM neuron cell [1]. This paper uses cell/neuron/HTM neuron interchangeably from this point onwards. The neuron model of the HTM is depicted in Fig. 1(c). A neuron is connected to three types of dendrite segments, namely (i) proximal dendrite segment which receives inputs from the neurons of the lower layer, (ii) distal dendrite segment which comprises of synaptic connections with the other neurons in the layer above, and (iii) distal dendrite segment which comprises of the lateral connections with other cells in the same layer. The neurons are stacked one above the other to form a mini-column and a row of mini-columns form a single layer of the HTM. In our case, we include only a single layer of the HTM (Fig.1(d)). Therefore, the feedback connections from the layer above do not exist in the neuron model of ours. The lateral connections of the distal dendrites enable the HTM to learn the temporal relations in the data stream. This is the most critical aspect of the HTM that we leverage for detection and prediction. Each HTM neuron can be in three possible states similar to biological neuron, namely (i) inactive, (ii) predictive state, and (iii) active state. The default state of a neuron is inactive.

3.1.2 Encoder

We denote the sequence of incoming data by $(X_1, X_2....X_T)$, where $X_i \in \mathbb{R}^m, i \in \{1, 2, ..., T\}$, i denotes the time index and m is the dimension of the signal. The encoder converts the signal at each time instant i into a sparse distributed representation (SDR), which is a high dimensional binary representation of size n (Fig. 1). SDRs are sparse representations where only a few bits are active for any input. In HTM this is typically set to 2% of the total size of the SDR that gives a good accuracy with a sparse representation of the incoming data. As a result, only fewer active bits overlap across different inputs. This is in contrast to dense representations, where many active bits can overlap.

3.1.3 Spatial Pooler

The next step is the spatial pooler. The spatial pooler computes a second SDR, which is the activation state of the mini-columns, of the same size as the input SDR. Each mini-column of an HTM is connected to a subset of the bits of the input SDR through synaptic connections collectively called as proximal dendrite segment. Typically, each neuron of the mini-columns is connected to a large fraction of the bits of the input SDR (50%). Initially, the bits are randomly selected and could be fixed for the rest of the time. All the neurons in a single mini-column share the same proximal synapses. Each synapse has a permanence value, which determines whether a connection is existent or not. This value can be incremented or decremented to create new synaptic connections or remove existing synaptic connections. A synaptic connection is said to be active if the input to the connection is one. The mini-columns are rank-ordered based on the number of active connections. The mini-columns that are activated are a certain number of columns from the top of this ordered list. The activation state of the mini-columns is the output of the spatial pooler, and thus the output of the spatial pooler is also an SDR.

3.1.4 Temporal Pooler

The output SDRs from the spatial pooler are given as inputs to the temporal pooler. The temporal pooler consists of multiple mini-columns, and each mini-column has a fixed number of HTM neurons stacked upon one another. Multiple mini-columns are stacked side by side to form a cortical column (Fig. 1). Each neuron of the mini-columns can comprise a minimum of two and sometimes up to a dozen distal dendrite segments. Each distal dendrite segment has synaptic connections that originate from multiple cells of the neighboring mini-columns in the same layer. The synaptic connections capture the temporal relations and constitute the temporal memory of an HTM. If there is an active synaptic connection between two cells of different mini-columns in the same layer, then a temporal relation exists between those cells. An active synaptic connection means that the cell from where the synaptic connection is originated is active. If the sum of the active synapses in any of the dendrite segments exceeds a certain threshold, then the cell enters the predictive state. The predictive state of a cell also provides the temporal context for the activation decision in the next time step.

The synaptic connections that present among different cells of the nearby mini-columns have weights. While learning, the weights of the inter-column synaptic connections are adjusted depending on the activation state of the cells in the predictive state in the next time step. If the predictive state in the current time step and the activation state in the next time step overlap, then it is taken to indicate that the temporal relation represented by the active synapses in the previous time step is correct. In such a case, weights of the active synaptic connections that correctly identified the predictive state are strengthened, and those that incorrectly identified or failed to identify are weakened. This is a Hebbian type learning and allows the HTM to learn a higher-order temporal representation of the sequential data, which can be used for prediction and detect anomalies.

3.2 Activation and Learning

3.2.1 Activation

Let's denote the current activation state of cells in a particular layer at time t by A^t (a $M \times N$ matrix where M is the number of cells per mini-column and N is the number of mini-columns in the layer), where $a^t_{i,j}$ is the i,jth element of A^t and denotes the activation state of cell i in column j. Let's denote a distal dendrite segment by d. Let the weight of the synapses of the dth segment of the ith cell of jth column be $D^d_{i,j}$. We note that only the weights of the synapses which are above a certain threshold are considered to be valid as a synaptic connection. The matrix of the established connection weights is denoted by $\tilde{D}^d_{i,j}$. The entries corresponding to the weights below the threshold are set to be zero in $\tilde{D}^d_{i,j}$.

Denote the predictive state of a neuron (i,j) by $\pi_{i,j}$. The neuron (i,j) is in a predictive state provided the sum of active synapses of at least one of the distal segments exceeds a certain threshold of θ_d. Thus, $\pi_{i,j}$ is given by,

$$\pi^t_{i,j} = \begin{cases} 1; & \text{if } \exists_d \, ||\tilde{D}^d_{i,j} \circ A^t||_1 > \theta_d, \\ 0; & \text{otherwise.} \end{cases} \quad (1)$$

where $\circ$ denotes the element-wise multiplication operation. Finally, only the cells of the mini-columns that were in the predictive states at time $t-1$ are activated. The activated cell is the cell of the active mini-column that was in the predictive state. The other cells in the mini-column are inhibited. This inhibition accounts for the specific temporal context as determined by the predictive states of the neurons in the mini-column. If none of the cells of an active mini-column are in a predictive state, then all the cells are activated. Let's denote the set of activated columns by the spatial pooler by C^t_a. Then the activation of a neuron (i,j) is given by,

$$a^t_{i,j} = \begin{cases} 1; & j \in C^t_a \text{ and } \pi^{t-1}_{i,j} = 1, \\ 1; & j \in C^t_a \text{ and } \sum_i \pi^{t-1}_{i,j} = 0, \\ 0; & \text{otherwise.} \end{cases} \quad (2)$$

3.2.2 Learning

The learning is a Hebbian type learning [22]. The learning algorithm only updates the weights of the synaptic connections of the cells that were in the predictive state or became active. If the predictive state of a neuron at the previous time step overlaps the activation state of a neuron at the current time step, then it is taken to indicate that the temporal relations captured by the synaptic connections are correct. The learning algorithm reinforces the temporal relations represented by the active synaptic connections. This results in the correct prediction. The learning algorithm also reduces the strength of the temporal relation represented by the inactive synaptic connections that failed to predict. Hence, the weights of the active synaptic connections and the weights of the inactive synaptic connections of a cell are increased and decreased respectively. Formally, the weights $D^d_{i,j}$ of the distal segment d of cell (i,j) that became active at time t are changed by the adaptation rule given by,

$$\Delta D^d_{i,j} = r^+ \hat{D}^d_{i,j} \circ A^{t-1} - r^- \hat{D}^d_{i,j} \circ (1 - A^{t-1}) \quad (3)$$

where r^+ and r^- are the increase and decrease rates of the permanence values of the synaptic connections. The synaptic permanence increment and decrement rates (i.e., r^+ and r^-) are set to 0.1. The matrix $\hat{D}^d_{i,j}$ is the matrix of weights with positive entries,

$$\hat{D}^d_{i,j} = \begin{cases} 1; & \text{if } D^d_{i,j} > 0, \\ 0; & \text{otherwise.} \end{cases} \quad (4)$$

If a column becomes active and no neuron in the column was predicted to become active in the previous time step, then the neuron with the most activated segments is picked and updated as above.

If the neuron that was in the predictive state does not become active, then it indicates that the active lateral connections that resulted in the predictive state represent an incorrect temporal relation. So the active lateral segments of the cells that were in the predictive state but remained inactive are decreased at a rate of r^-_f. The synaptic permanence decrement for predicted inactive segments, r^-_f is set to 0.01 such that $r^-_f \ll r^-$. Formally, the weights of these active segments are decreased by the adaptation rule given by,

$$\Delta D^d_{i,j} = -r^-_f \hat{D}^d_{i,j} \text{ where } a^t_{i,j} = 0 \text{ and } ||\tilde{D}^d \circ A^{t-1}||_1 > \theta_d \quad (5)$$

In summary, the learning algorithm described here learns a temporal representation of the sequential data by adjusting the weights of the lateral connections between the cells. The adjustments of the weights are based on the correctness of the temporal relation represented by the synaptic connections.

3.2.3 Time complexity of the HTM

Here we discuss the time complexity of the three operations: encoder, spatial pooler, and temporal pooler.

The encoder converts the input data into a sparse vector. The encoder used in our model is a scalar encoder (refer to Table 1). A bit is turned on in a scalar encoder if the value of the input falls within the window of the values the bit is associated with. Essentially, this operation entails checking which window the input value falls within. This requires time complexity of $O(W)$, where W is the number of windows the range of values are represented by. The maximum number of windows is limited by the size of the encoder. Therefore, the overall time complexity of the encoder is $O(n)$ where n is the size of the encoder.

The spatial pooler converts the sparse vector computed by the encoder into a second sparse vector as outlined earlier. Each bit of the output of the spatial pooler is connected to as many as 50% of the bits of the input sparse vector. Let's denote the size of the sparsity by w (i.e., $w < 2\%$), which denotes the number of bits that will be active among the n bits of a vector and is typically a small number. First, the spatial pooler computes the number of proximal synaptic connections that are active for each bit in the output of the spatial pooler. For each bit, this is an $O(w)$ operation because the pooler only needs to check which of the active bits of the input SDR are connected to this particular bit in the output of the pooler. Therefore, the operation of computing the number of active connections for each of the n bits is in total $O(nw)$. Next, the spatial pooler orders the bits in descending order of the number of active connections and the top k bits are chosen. This is an $O(n \log n)$ operation. Therefore, the overall time complexity of the spatial pooler is $O(n \log n) + O(nw) \approx O(n \log n)$ (since $w \ll n$). Also,

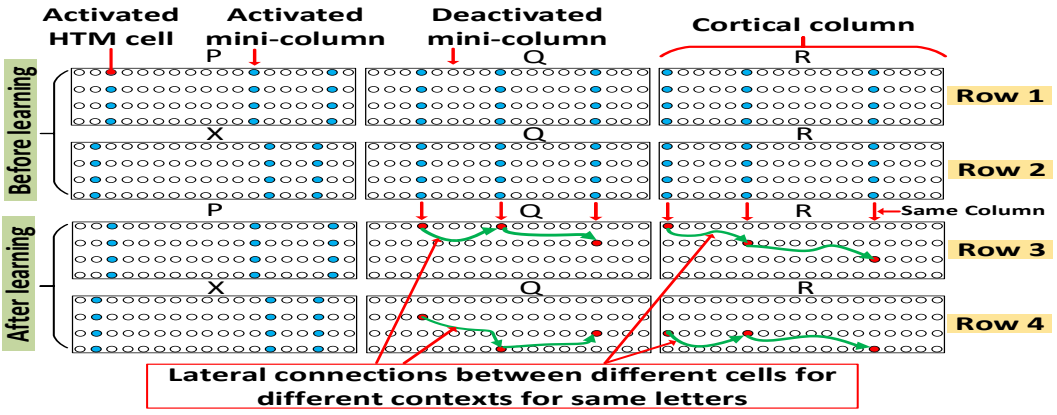

Fig. 2: Illustrative example of learning two different sequences: 'PQR' and 'XQR'.

the final activation step described by Eqn. 2 is just $O(wM)$, because this step is the computation to decide which of the M neurons of each of the w active mini-columns are to be activated.

The temporal pooler computes the predictive state of the neuron cells in the mini-columns as given by Eqn. 1. This operation computes $\tilde{D}^d_{i,j} \circ A^t$ for each of the d segments of a neuron cell. For each segment the computation $\tilde{D}^d_{i,j} \circ A^t$ is an $O(w)$ operation. Therefore, when repeated for the d segments, it is an $O(dw) \approx O(w)$ operation. The predictive state is set based on this computation as described in Eqn. 1. This is repeated for every neuron; therefore, the time complexity of the temporal pooler is $O(nMw) \approx O(n)$.

3.3 An illustrative example

In this section, we discuss an example that illustrates how the HTM learns to represent multiple temporal sequences even if there are overlaps between the sequences. The example is shown in Fig. 2. An HTM response for two sequences 'PQR' and 'XQR' is shown in Fig. 2. Time advances from left to right in all the rows. Each ○ in Fig. 2 represents an HTM cell and these cells (in this case 4 cells) are stacked one on top of another to form a mini-column. Each region of the HTM contains 18 mini-columns, called as cortical column. The top two rows depict the cell firings before learning the temporal relations, and the bottom two rows depict the cell firings after learning the weights of the lateral connections.

The response of the HTM before learning is shown in the top two rows. Before learning (the top two rows), the lateral synaptic connections between the neighboring cells (i.e., green connection in Fig. 2) have not been learned yet, and so none of the cells are driven to be in the predictive state. As a result, all the cells (i.e., blue circles for activation in Fig. 2) in an active mini-column are activated. We also note that only a very small fraction of the mini-columns are activated for every input.

After the lateral weights are learned (the bottom two rows), we observe that the same mini-column is invoked for the same letter but only one cell is activated in a mini-column this time. Though the cells activated are of the same mini-columns, the positions of the cells active in the same mini-columns are different. For example, the positions of the activations in the mini-columns that correspond to 'Q', shown in row-3, is completely different from the positions of the activations in row-4 for 'Q'. This is because the temporal context is different for the two scenarios depicted in row-3 and row-4 where in row-3 'Q' follows 'P' and in row-4 'Q' follows 'X'. For the same reason, the positions of the activations for 'R' are different in row-3 and row-4. Though 'R' follows 'Q' in both cases, the starting letters in the sequence are different ('P' and 'X') in the two cases. Therefore, the sparse encoding of 'R' not only depends on 'Q' but also on 'P' or 'X'. This clearly shows that the HTM learns higher-order overlapping temporal sequences.

It follows that if a temporal sequence is different but contains a letter that overlaps with another sequence, then a different cell of the mini-column corresponding to the common letter is activated for the two sequences. Thus, the different lateral connections responsible for activation for the different cells are strengthened over time. This allows the HTM to learn different temporal sequences even when there are overlaps between the sequences. *We emphasize that the column structure with multiple cells is the key structural aspect that allows the HTM to learn multiple temporal sequences.*

3.4 Source of μPMU dataset

To demonstrate anomaly detection and simultaneous multi-step prediction, we use *two open-source*, and real-time current magnitude datasets collected from the μPMU sensors installed at the Lawrence Berkeley National Laboratory's (LBNL) distribution grid [23]. This is the first μPMU network installed on a real electrical grid for research purposes, and the datasets collected from this network are the only available open-source datasets on real-time μPMU data, to the best of our knowledge [24]. We randomly pick two different datasets from the available LBNL's datasets to test the HTM. The first dataset (i.e., dataset 1) is collected from the *a6 bus 1* located in a 7.2 kV grid, and the second dataset (i.e., dataset 2) is collected from the low side of a 1500 kVA delta/wye transformer with a 480V/208V rating. The part name of the μPMU used to collect the two datasets is *PQube3*, which can output at 120 Hz frequency. Since data is collected in millisecond resolution and almost all practical events happen at a larger time scale [9], the raw data is resampled at 1-sec interval for 12 days and 13 hours (1 Million+ data points). In the next section, we use the two datasets to show real-time anomaly detection in smart grids.

4 Demonstration of Anomaly Detection

The anomalies in a smart grid can be classified into two main categories: voltage or current magnitude variation and frequency variation. A voltage sag (a short term low

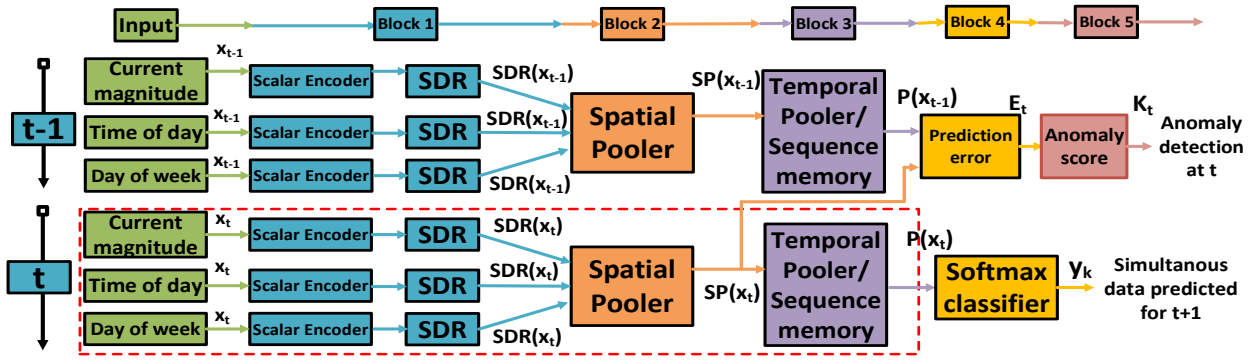

Fig. 3: HTM model for anomaly detection and simultaneous data prediction.

voltage), a voltage or current spike (a short term high voltage or current above 110% normal value), a brownout (reduced voltage for an extended period), an overvoltage or overcurrent (an extended period of high voltage or current), etc., can be categorized as voltage or current magnitude anomalies. Frequency deviation from the normal limit can be categorized as a frequency variation. As noted earlier, μPMU sensors capture voltage or current magnitude and frequency information at ms timescales. In this section, we demonstrate that the HTM can be used to detect anomalies in μPMU data in real-time, while learning from just one-pass, and in an unsupervised fashion. We also demonstrate that learning is continuous, and by doing so, we show that the HTM model can potentially adapt to concept drifts.

4.1 HTM implementation for anomaly detection

The architecture of the HTM model for real-time anomaly detection and simultaneous data prediction is presented in Fig. 3. In this section, we only focus on the HTM implementation for anomaly detection, which has the following set of blocks. The first block is the scalar encoder that converts the data x_{t-1} at time step $t-1$ into a sparse distributed representation denoted by SDR(x_{t-1}) [25]. The second block is the spatial pooler [26] that transforms each SDR(x_{t-1}) matrix into a sparse binary vector representation, SP(x_{t-1}) [1] (see Section 3.1). The final block is the temporal pooler, which generates P(x_{t-1}). The term P(x_{t-1}) is the prediction of SP(x_t) based on SP(x_{t-1}) at $t-1$, as described in Section 3.1. The final block computes the error between the actual value, SP(x_t) and the predicted value, P(x_{t-1}) (computed at the previous time step) as given by [17],

$$E_t = 1 - \frac{SP(x_t) \circ P(x_{t-1})}{|SP(x_t)|} \tag{6}$$

where E_t is known as the prediction error at time-step t and $\circ$ is the dot product. $|SP(x_t)|$ is the modulus of the binary vector $SP(x_t)$. To assess how large the deviation E_t is, we compute a distribution of the deviations over a local window of time Δt. To compute this distribution, we compute the mean and standard deviation of the variables $E_{t'}$ $(t' \leq t)$ over the window of time Δt that ends at the current time t. Denote the mean and standard deviation of $E_{t'}$ over this window Δt by μ_t and σ_t. Then μ_t and σ_t are given by,

$$\mu_t = \frac{\sum_{i=0}^{\Delta t-1} E_{t-i}}{\Delta t}, \tag{7}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{\Delta t-1} (E_{t-i} - \mu_t)^2}{\Delta t - 1}. \tag{8}$$

Then the anomaly score K_t is calculated using the Q function [27] that is given by,

$$K_t = 1 - Q\left(\frac{\mu_t^p - \mu_t}{\sigma_t}\right), \tag{9}$$

$$\text{where } \mu_t^p = \frac{\sum_{i=0}^{p-1} E_{t-i}}{p}. \tag{10}$$

Here the term μ_t^p is the calculated mean over a shorter time window, p. The value K_t is large if the value Q for μ_t^p is small, i.e., if the probability that a deviation is greater than μ_t^p is small. Hence, if K_t is greater than a certain threshold, the algorithm concludes that the probability of occurrence of an anomalous event is high and declares the current state to be anomalous.

Table 1 shows the parameter values set for the HTM in the anomaly detection problem for both datasets. The same values are reused for the prediction problem in Section 5 of the paper. We choose this parameter setting because it was shown to work well for a wide range of datasets in other applications [17].

4.2 Capturing temporal and spatial anomalies

In smart grids, the current magnitude data from μPMU may have a significant amount of spatial and temporal fluctuations. The spatial fluctuations in voltage or current arise in real-time from sudden or unpredictable switching of large loads, sudden power outage in the same or nearby locality, or any sudden grid line faults. Temporal or contextual fluctuations are variations that occur due to peak or off-peak hours of a day, day or night cycle, weekend or weekdays cycle, seasonal variations (temperature, humidity, etc.), etc. The two fluctuation types are of different nature. Spatial fluctuations are independent of any temporal context, whereas temporal fluctuations are dependent on specific contexts that are temporally extended. This makes the problem of detecting temporal anomalies challenging because learning higher-order sequences are harder.

In the experiments for both datasets, we feed data into the HTM one at a time, and a *small initial amount of data* (i.e., first 1 hour 23 minutes or first 5000 data points of 1 million+ data points) is used for initial training of the temporal pooler of the HTM (the initial gray region in Fig. 4). In this period, only the training is switched on and anomaly detection is switched off.

We demonstrate the effectiveness of the HTM in detecting spatial as well as subtle temporal anomalies present in the current magnitude of the μPMU data by using the dataset 1 in Fig. 4. In Fig. 4, we find that the HTM identifies

TABLE 1: Parameter setting

Time of day	Day of week	Current magnitude	Spatial Pooler	Temporal Pooler
Encoder type: Scalar	Encoder type: Scalar	Encoder type: Scalar	Column count, N: 2048	Column count, N: 2048
Maximum value: 60	Maximum value: 7	Maximum value: 40	Global inhibition: 1	Cells/Column: 16
Minimum value: 0	Minimum value: 0	Minimum value: 0	Seed: 1956	Max. synapses/segment: 32
Total bits, n: 600	Total bits, n: 100	Total bits, n: 109	Synaptic perm. con.: 0.5	New synapse count: 32
Total active bits, w: 29	Total active bits, w: 29	Total active bits, w: 29	Synaptic perm. act.: 0.0001	Synaptic perm. inc./dec.: 0.1

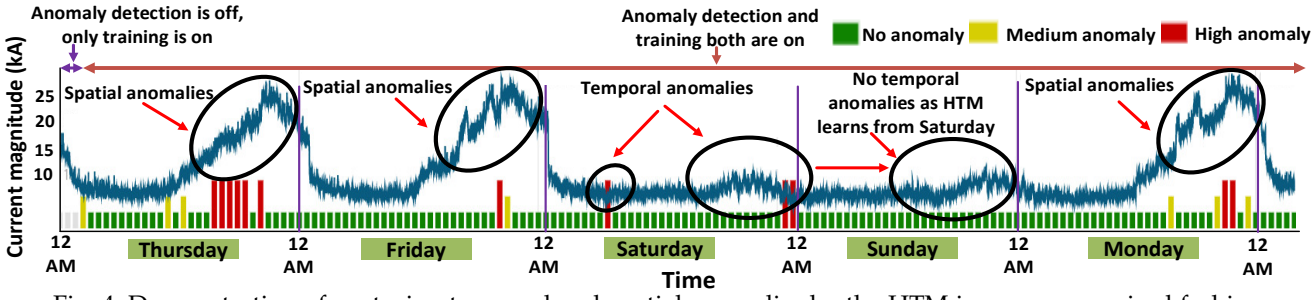

Fig. 4: Demonstration of capturing temporal and spatial anomalies by the HTM in an unsupervised fashion.

six instances on *Thursday* night as anomalies (indicated by red bars). The HTM does so because the variations on Thursday night are being observed for the first time. The HTM learns this pattern after the first observation because we find that the HTM does not label most of the instances on the next occurrence of a similar pattern, which is on *Friday* night, as anomalous. Next, we observe that the HTM identifies two instances on *Saturday* night as anomalous. This is again because the pattern on Saturday night is a new pattern that has not been observed on the previous days, i.e., on Thursday and Friday. This pattern is a temporal anomaly because it is a pattern that is typically observed on the weekends and not due to any spatial fluctuation. This suggests that the HTM identifies temporal anomalies. We also find that the HTM correctly identifies anomalies on Monday night. These are spatial anomalies because the pattern observed on Monday night shown here is a deviation from the pattern on the previous Thursday and Friday night and possibly is the result of a spatial fluctuation. Overall, the last two observations are suggestive that the HTM can identify both temporal and spatial anomalies.

4.3 Continuous online unsupervised learning

The challenge for the HTM is to learn the temporal patterns that are repetitious so that they are not wrongly identified as anomalies. It is clear that the weekends and the weekdays have different current magnitude patterns. In Fig. 5, we demonstrate that the HTM does not identify any instance of the temporal pattern of the weekends on the second occurrence of the weekend as anomalous. We use the dataset 1 for this demonstration. This demonstration suggests that the HTM has learned the higher order temporal patterns that occur on the weekends on the first observation itself. This indicates that the HTM can learn in a *continuous-online fashion*, which in turn is suggestive that it can account for *concept drift*. We also emphasize that the learning in HTM is without any human intervention and manual parameter tweaking, i.e., in an *unsupervised fashion*.

4.4 Comparison with other anomaly detection algorithms

Anomaly detection algorithms can be either supervised or unsupervised. Supervised algorithms require labeled data and periodic retraining with changing conditions. The data presented here is unlabelled and so supervised learning approaches are not considered for comparison. Unsupervised algorithms can be of different types, such as simple or dynamic thresholding, complex statistical models, distant based methods, etc.

This paper considers five state-of-the-art real-time, unsupervised algorithms, namely Windowed Gaussian (i.e., dynamic thresholding), Random Cut Forest (i.e., distance-based models), Bayesian Changepoint, EXPoSE, Relative Entropy (i.e., all three are complex statistical models) to compare with the HTM, and several of their properties are shown in Table 2. This table indicates that the HTM can learn in one-pass and unsupervised fashion, detect spatial and temporal anomalies, and predict future observations in real-time. Moreover, the table also indicates that the HTM has a slightly higher time complexity compared to the other algorithms (except Random Cut Forest). The reason behind this is that the HTM learns long-term different temporal sequences even when there are overlaps between the sequences in its sparse sequential memory. The learning happens by adjusting the weights of the lateral connections between the cells. The slightly higher time complexity of the HTM does not hamper its real-time anomaly detection capability in smart grids that is discussed in Section 4.4.2.

4.4.1 *Numenta Anomaly Benchmark (NAB)*

We use the Numenta Anomaly Benchmark (NAB) [20] to compare the five algorithms with the HTM. Regular scoring methods, such as precision and recall cannot be used for scoring as they are not suitable for real-time problems. The NAB scoring mechanism has been designed based on what a good real-time detection algorithm should be able to do: *detect all anomalies in a streaming data, in real-time, with less false alarms, and in an automated fashion*.

The scoring mechanism contains three components: anomaly windows, application scoring profiles, and a scoring function. Anomaly windows are ranges of data points that surround each anomalous instances. The NAB score accounts for the differences in the importance of false positives and false negatives in applications by considering three different application dependent scoring profiles: (a) Standard, (b) Reward few false positives, and (c) Reward few false negatives.

TABLE 2: Comparison among state-of-the-art real-time anomaly detection algorithms

Algorithms	Unsupervised	Noise immunity	Spatial anomaly	Temporal anomaly	Online learning	Non parametric[2]	Multi-step prediction	Time complexity[4]
HTM [28]	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	$O(n \log n)$
Random Cut Forest [29]	Yes	No	Yes	No	Yes	No	No	$O(D\, log \frac{\|n\|}{L_1(u,v)})$
Bayesian Changepoint [30]	Yes	Yes	Yes	No	Yes	No	No	$O(n)$
Windowed Gaussian [31]	Yes	No	Yes	No	No	No	No	$O(nw^2)$
EXPoSE [32]	Yes	Yes	Yes	Yes	Yes	Yes	No [3]	$O(n)$
Relative Entropy [33]	Yes	Yes	Yes	Yes	Yes	Yes	No [3]	$O(n^\alpha), \alpha < 3$

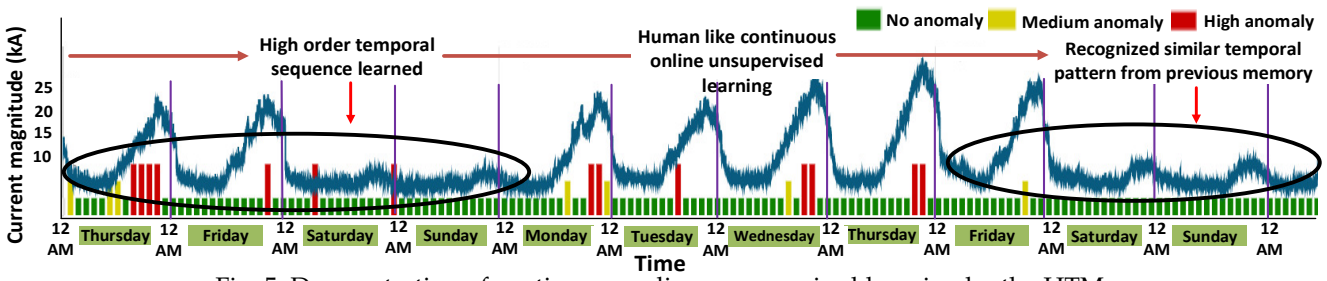

Fig. 5: Demonstration of continuous online unsupervised learning by the HTM.

The scoring function is such that an anomaly detected within the anomaly window is considered as true positive and given a positive score (e.g., Point 1 is given a score of +0.98 in Fig. 6). An anomaly detected outside the anomaly

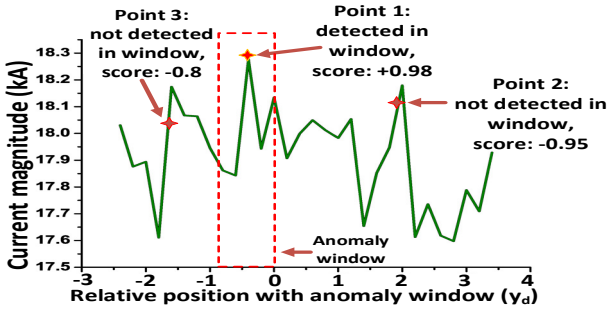

Fig. 6: NAB window and scoring process.

window is considered as false positive and given a negative score, which is also scaled depending on the position relative to the window. (e.g., Point 2 is given -0.95, and Point 3 is given -0.8 in Fig. 6). The final scores for a detection d is calculated using the sigmoidal scoring function given by,

$$S^A(d) = (A_{TP} - A_{FP}) \left(\frac{1}{1 + e^{5y_d}} \right) - 1, \qquad (11)$$

where A is the application profile under consideration (A $\subset$ {Standard, Reward few false positives, Reward few false negatives}), y_d is the relative position within the anomaly window, and A_{TP}, A_{FP} are weights which depend on the application profile A. Missing to detect any anomaly is considered as a false negative and is assigned a score of A_{FN}. The total score, TS^A over a dataset is calculated as,

$$TS^A = \left(\sum_{d=1}^{D} S^A(d) \right) + A_{FN} \times N_f, \qquad (12)$$

where N_f is the total number of false negatives, and D is the total number of detected anomalies in the dataset.

4.4.2 *Calculation of NAB score with latency time*

Calculation of NAB score requires the ground truth information. A total of 15000 *unbiased anomaly points* have been identified and labelled as anomalies for both datasets. The calculated NAB scores for both datasets and the detection latency times for all the methods are shown in Table 3.

TABLE 3: NAB score board for both μPMU datasets

Algorithms	Latency (ms)	Standard		Reward few false positive		Reward few false negative	
		set 1	set 2	set 1	set 2	set 1	set 2
HTM	7.8	**87%**	**89%**	**83%**	**86%**	**89%**	**90%**
Random Cut Forest	19	15%	18%	12%	16%	34%	38%
Bayesian Changepoint	2.8	74%	72%	72%	70%	77%	80%
Windowed Gaussian	3.6	65%	69%	61%	65%	69%	71%
EXPoSE	2.1	67%	69%	63%	66%	71%	73%
Relative Entropy	0.5	20%	24%	21%	26%	20%	26%

Table 3 shows that the HTM achieves a far-better score than other real-time anomaly detection algorithms in terms of detection accuracy for both datasets. However, the HTM has slightly higher detection latency (i.e., 7.8 ms) compared to other algorithms (except Random Cut Forest). The reason behind this is that the HTM has a slightly higher time complexity (see Table 2) than others because of its sequence learning capability in the complex sequential memory. The detection latency times in Table 3 are calculated in a 4.4 GHz Intel Core i9 processor with 16 cores and 32 GB of RAM. As the available voltage/current frequency in the smart grid is 50/60 Hz (i.e., period 20/16 ms), we want to emphasize that the detection latency of the HTM (i.e., 7.8 ms) is *low enough* to detect anomalies in smart grids in real-time. More precisely, the HTM is capable of detecting anomalies in less than half cycle time (full cycle = 20/16 ms, half cycle = 10/8 ms) in smart grids with good accuracy compared to other real-time anomaly detection algorithms. Moreover, the HTM model can be used for simultaneous

2. Non-parametric refers to no requirement of application specific hyper-parametric tuning.

3. Relative entropy based methods have been used for forecasting in economics [34], and methods like ExPoSE have been used for one-step prediction.

4. Notation: n = point set; w = sparsity; D = dimension; L_1 = Manhattan distance; $(u, v) \subset n$.

multi-step prediction (Table 2). Hence, we conclude that the HTM is competitive and has more capability than the other methods. This is primarily because the HTM *learns a general representation that can be used for multiple tasks* (in our case, prediction). We demonstrate this in the next section.

5 Demonstration of Multi-step Prediction

The smart grid is stochastic and dynamic in nature and predicting system behavior in real-time is critical from the point of system control. Though load forecasting [35] is a widely studied problem, in this section we focus on a specific prediction problem, namely short-term prediction (say 5 minutes ahead prediction). The real-time prediction is important from the point of view of control and planning of the grid operation. For example, it may be beneficial to predict a sudden change in power in the smart grid in real-time. This prediction can be used to respond early to compensate for this sudden change by regulating power flow to the affected part of the smart-grid. A common regulation method is to dynamically adjust the *governor* set point [36] of the generators. The set point adjustment can be made more reliable by taking into account the predictions of the transitions of the state of the grid. Here, we demonstrate that the same HTM that was trained for anomaly detection can be reused for predicting future observations in real-time.

5.1 HTM implementation for multi-step prediction

We have argued and shown that the temporal pooler of the HTM learns higher-order temporal sequences of the observed data in one-pass fashion. Therefore, the same HTM model can be *used for multi-step prediction in real-time* by adding a *softmax classifier* at the output of the temporal pooler. The softmax classifier outputs the class of the predicted state by the temporal pooler. The architecture for multi-step prediction is shown in Fig. 3. For classification, the full range of the possible current magnitude values is divided into 22 disjoint classes (k). The classifier block is a single layer of a fully connected feed-forward network with the number of output neurons same as the number of classes. If the jth output neuron of the feed-forward network is given by a_j and the ith output of the temporal pooler is given by $P(x_t)_i$, then a_j is given by,

$$a_j = \sum_{i=1}^{n} \theta_{ij} P(x_t)_i, \tag{13}$$

where θ_{ij} is the weight of the connection from the ith neuron of the temporal pooler, $P(x_t)_i$, to the jth neuron of the feed-forward network, a_j, and n is the dimension of $P(x_t)_i$. The probability y_k of the predicted data falling into class k is calculated by the softmax function as follows:

$$y_k = \frac{e_k^a}{\sum_{i=1}^{k} e_i^a}. \tag{14}$$

The weights θ_{ij} are updated by the descent along the gradient of the least-squares error of the prediction. If the update of each weight θ_{ij} is denoted by $\Delta\theta_{ij}$, the term $\Delta\theta_{ij}$ can be expressed as follows:

$$\Delta\theta_{ij} = -\lambda(y_j - z_j)P(x_t)_i, \forall i, j, \tag{15}$$

where z_j is the observation and λ is the learning rate. As $P(x_t)_i$ is highly sparse, only a small portion of the weights are updated at any time and this results in faster prediction. The parameter settings of the HTM for the multi-step prediction is already listed in Table 1.

5.2 Multi-step prediction on μPMU data using the HTM

Fig. 7 demonstrates 5 min. ahead prediction on μPMU data for dataset 1. In this demonstration, the probabilities of all predicted data-classes are calculated using Eqn. 14. The value having the highest probability is chosen as the final predicted value. We note that *the prediction is in real-time.* The 'red' colored curve is the predicted curve and is shifted 5 time-steps to increase the visibility of the comparison between the actual and the predicted points. In all the circled regions of Fig. 7, the predicted spikes ("red") follow an actual spike ("black") indicating that the HTM is able to predict the fluctuations or glitches. The predicted "red" curve may not exactly match the "black" curve point by point but we observe that the prediction by the HTM is able to capture the variations observed in the data. To prove this claim, in the next section, we quantify the HTM's performance using two Key Performance Indicators (KPIs) and compare it with other sequence prediction algorithms.

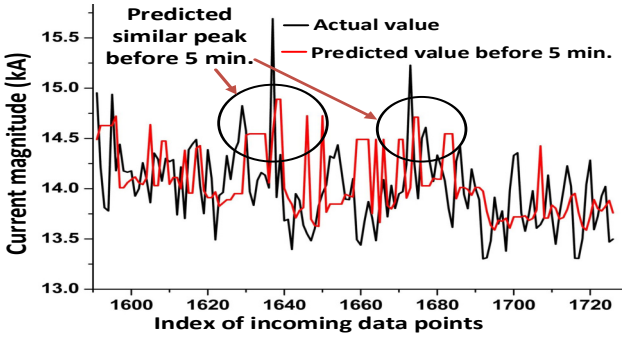

Fig. 7: Current magnitude prediction 5 min. ahead.

5.3 Comparison with other multi-step prediction algorithms

This section compares six state-of-the-art sequence prediction models, namely Adaptive Filter, Time Delayed Neural Network (TDNN) and four versions of LSTM with the HTM.

Adaptive Filter [37] is a self-learning predictive model that can adapt quickly in real-time. This model is implemented by using the LMS algorithm with a filter size of 10. TDNN [38] and LSTMs are widely used *state-of-the-art predictive models that require batches/mini-batches of data* for training. Here, these models are made adaptive and partially online by retraining them at different time-steps using different ranges of past data points. TDNN is implemented with 100 input units, 1 hidden layer with 200 units, 1 output unit and retrained after every 336 time-steps using the last 3000 data points.

Here, we consider four versions of the LSTMs, namely LSTM-Online, LSTM-1000, LSTM-3000, and LSTM-6000. They are implemented with 3 input units, 20 LSTM units, and 1 output unit. LSTM-Online is retrained at every time-step using the last 100 data points, whereas LSTM-1000, LSTM-3000, and LSTM-6000 are retrained at every 1000th time-step using the last 1000, 3000, and 6000 data points, respectively. For retraining, different time intervals and different ranges of data points have been selected to illustrate the effectiveness of the HTM as a one-pass learner. Though

the performance of LSTM is observed to improve with the increase in the data points for retraining, we emphasize that the HTM is able to achieve the same or even better performance just by learning from one-pass over the data.

The qualitative comparisons of the algorithms are given in Table 4. The comparison is clearly suggestive that the HTM is effective in capturing long term dependency while learning from just one-pass over the data. Moreover, the HTM can quickly adapt to concept drifts without much parameter tuning compared to other methods.

TABLE 4: Comparison among state-of-the-art sequence prediction algorithms.

Algorithms	One-Pass Learning	Time to Adapt	Long Term Dependency	Non Parametric
HTM [39]	**Yes**	**Short**	**Yes**	**Yes**
Adaptive Filter [40]	Yes	Short	Limited	No
TDNN [41]	No	Long	Limited	No
LSTM-Online [42]	Yes	Long	Yes	No
LSTM-1000 [42]	No	Long	Yes	No
LSTM-3000 [42]	No	Long	Yes	No
LSTM-6000 [42]	No	Long	Yes	No

To quantitatively illustrate the multi-step prediction accuracy of the HTM, we compare the HTM with other algorithms using two KPIs, namely Normalized Root Mean Square Error (NRMSE) and Negative Log-likelihood (NLL), on both datasets. NRMSE is sensitive to outliers and low NRMSE value indicates an almost pointwise perfect fit between the true and the predicted values. The NRMSE is calculated as follows:

$$NRMSE = \sqrt{\frac{\sum_{t=1}^{N_d}(Pred_t - True_t)^2}{\sigma}} \tag{16}$$

where σ is the standard deviation of the actual data, N_d is the total number of data points, and $Pred_t$ is the predicted value of a true value $True_t$ at time t.

The NRMSE score only considers the point-wise prediction error. To measure the accuracy of prediction of a sequence we consider an alternate measure, the Negative Log-likelihood (NLL). The NLL score is computed using the probability function $P(y_t|y_1.....y_{t-1})$. This function captures the dependency of the prediction y_t on the predicted values at the previous time-steps, which are $y_1, ...y_{t-1}$. The NLL score is simply the negative of the log of this probability:

$$NLL = -\frac{1}{N_d}\sum_{t=1}^{N_d} log(P(y_t|y_1.....y_{t-1})), \tag{17}$$

where N_d is the total number of data points. Here, the lesser NLL score indicates more accurate sequence prediction.

Fig. 8(a) shows that HTM, TDNN, Adaptive Filter, and LSTM-Online have a similar and a lower NRMSE than LSTM-1000, LSTM-3000, LSTM-6000 models for both datasets. We attribute this to the presence of frequent outliers in the predicted data points of the LSTM-1000, LSTM-3000, LSTM-6000 models. This suggests that HTM, TDNN, LSTM-Online, and adaptive filters are more accurate. Fig. 8(b) shows the NLL score for all the predicted data points. Adaptive filter and TDNN are not considered for the comparison because they have limited capability in capturing long term dependency (Table 4). The NLL score of the HTM

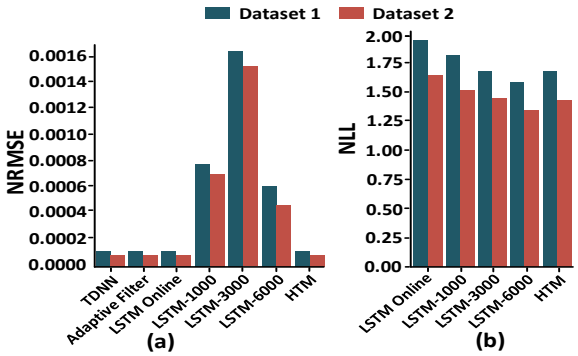

Fig. 8: Comparison of NRMSE and NLL values.

is better than the LSTM-Online, LSTM-1000 models and is similar to LSTM-3000. The NLL score of LSTM-6000 is slightly better than the HTM. We attribute this to multiple retraining with a larger set of previous data points, which is 6000 in this case. *However, the HTM is more effective because it achieves a similar score by learning from just one-pass, whereas LSTMs require multiple retraining over the same set of data points.*

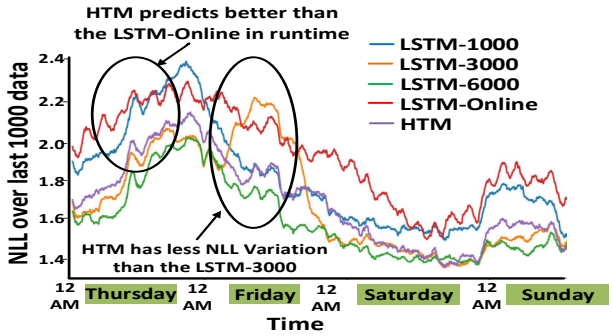

Fig. 9: Comparison of NLL value over last 1000 data points.

Fig. 9 shows the plot of NLL value computed over the last 1000 data points for dataset 1. It is clear from Fig. 9 that HTM, LSTM-3000, and LSTM-6000 show a variation of NLL that over time is lower than the LSTM-Online and LSTM-1000 models. *Though the HTM has the same NLL score as LSTM-3000 (Fig. 8(b)), it is observed to exhibit less NLL variation than LSTM-3000 overall (Fig. 9).*

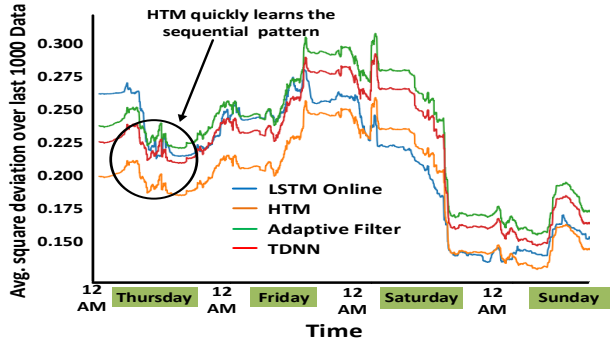

Fig. 10: Comparison of average square deviation over last 1000 data points.

Fig. 10 shows the variation of average square deviation error where each value is the average over the last 1000 data points. It is clear from Fig. 10 that the HTM initially quickly learns the sequential pattern and exhibits lower square deviation error than the LSTM-Online, Adaptive Filter, and TDNN. Later on, we observe that the HTM

consistently exhibits similar or lower square deviation error compared to the other methods. This proves that the HTM learns faster than the LSTM-Online, Adaptive Filter, and TDNN. As LSTM-1000, LSTM-3000, and LSTM-6000 require retraining, they learn much slowly compared to the HTM; therefore, they are not considered here for comparisons.

We observe that the NLL in Fig. 9 and the square deviation error in Fig. 10 have higher values on Thursday and Friday (i.e., weekdays) compared to Saturday and Sunday (i.e, weekends). The reason behind this is that the current magnitude is more stochastic on the weekdays compared to the weekends because the variation of loads on the weekdays is higher than the weekends. In other words, the loads on the weekends are more deterministic than the loads on the weekdays. We also observe that loads on Sunday are more stochastic than the loads on Saturday. Therefore, the NLL and the square deviation error on Sunday have slightly higher values than Saturday in Fig. 9 and Fig. 10.

6 Limitations and Future Work

The compelling feature of the proposed HTM is that it can detect anomalies and simultaneously predict future observations in real-time. Moreover, the HTM can learn in just one-pass and in an unsupervised fashion and can handle *concept drifts* efficiently. But the limitation is that the current implementation can not classify the anomalies based on the cause of the anomaly (e.g., a grid-line fault or a sudden load transient). Our future work is to extend the HTM model for real-time anomaly classification and simultaneous data prediction not only in smart grids but also in other CPSs applications [43], [44].

7 Conclusion

μPMU sensors sample the grid voltage/current waveform in an ultra-precise and synchronized fashion. Therefore, they can support many diagnostic applications, such as anomaly detection and simultaneous data prediction in real-time. In this paper, we introduced a neuro-inspired architecture called the HTM and discussed its structure and cortical learning algorithm. The HTM learns a general temporal sparse representation that can be leveraged for multiple tasks. The HTM can also be trained continuously, in one-pass and in an unsupervised fashion. This makes them suitable for real-time applications. In this work, we demonstrate how the HTM can be used for anomaly detection and simultaneous multi-step prediction in real-time. To demonstrate the effectiveness of the HTM, it is compared with five state-of-the-art real-time anomaly detection algorithms and six other state-of-the-art prediction algorithms. We showed that the HTM achieves competitive scores on both these tasks, while the other state of the art algorithms are either less accurate or can be used for one of the tasks only. *Moreover, the HTM achieves this performance by learning in just one-pass, and in an unsupervised fashion. We strongly believe that this work will serve as a motivation for research on neuro-inspired machine learning algorithms in the smart grid by demonstrating that such algorithms are more effective for real-time applications.*

Acknowledgments

The authors would like to thank the anonymous reviewers, and the associate editor for their valuable comments that greatly helped to improve this paper. This work was partially supported by the University of California, Office of the President under Grant No. LFR-18-548175, and by the NSF under Grant ECCS-1839429, ECCS-2028269 and CMMI-1739503. The views and conclusions contained in this paper are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the University of California, Office of the President, and NSF.

References

[1] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in neural circuits*, vol. 10, p. 23, 2016.

[2] R. Moghaddass and J. Wang, "A hierarchical framework for smart grid anomaly detection using large-scale smart meter data," *IEEE Transactions on Smart Grid*, vol. 9, no. 6, pp. 5820–5830, 2017.

[3] M. Zhou, Y. Wang, A. K. Srivastava, Y. Wu, and P. Banerjee, "Ensemble-based algorithm for synchrophasor data anomaly detection," *IEEE Transactions on Smart Grid*, vol. 10, no. 3, pp. 2979–2988, 2018.

[4] G. Napier, E. Davidson, S. McArthur, and M. J. McDonald, "An automated fault analysis system for SP energy networks: Requirements, design and implementation," in *2009 IEEE Power & Energy Society General Meeting*. IEEE, 2009, pp. 1–7.

[5] V. K. Singh and M. Govindarasu, "Decision tree based anomaly detection for remedial action scheme in smart grid using pmu data," in *2018 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2018, pp. 1–5.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[7] M. Landauer, M. Wurzenberger, F. Skopik, G. Settanni, and P. Filzmoser, "Dynamic log file analysis: an unsupervised cluster evolution approach for anomaly detection," *computers & security*, vol. 79, pp. 94–116, 2018.

[8] J. Valenzuela, J. Wang, and N. Bissinger, "Real-time intrusion detection in power system operations," *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1052–1062, 2012.

[9] Y. Zhou, R. Arghandeh, I. Konstantakopoulos, S. Abdullah, A. von Meier, and C. J. Spanos, "Abnormal event detection with high resolution micro-PMU data," in *2016 Power Systems Computation Conference (PSCC)*. IEEE, 2016, pp. 1–7.

[10] S. Brahma, R. Kavasseri, H. Cao, N. R. Chaudhuri, T. Alexopoulos, and Y. Cui, "Real-time identification of dynamic events in power systems using pmu data, and potential applications—models, promises, and challenges," *IEEE transactions on Power Delivery*, vol. 32, no. 1, pp. 294–301, 2016.

[11] J.-S. Chou and N.-T. Ngo, "Smart grid data analytics framework for increasing energy savings in residential buildings," *Automation in Construction*, vol. 72, pp. 247–257, 2016.

[12] F. Gao, J. S. Thorp, A. Pal, and S. Gao, "Dynamic state prediction based on auto-regressive (AR) model using PMU data," in *2012 IEEE Power and Energy Conference at Illinois*. IEEE, 2012, pp. 1–5.

[13] J. Sia, E. Jonckheere, L. Shalalfeh, and P. Bogdan, "Pmu change point detection of imminent voltage collapse and stealthy attacks," in *2018 IEEE Conference on Decision and Control (CDC)*. IEEE, 2018, pp. 6812–6817.

[14] H. Yang, A. Alphones, W.-D. Zhong, C. Chen, and X. Xie, "Learning-based energy-efficient resource management by heterogeneous RF/VLC for ultra-reliable low-latency industrial IoT networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5565–5576, 2019.

[15] H. Yang, X. Xie, and M. Kadoch, "Machine Learning Techniques and A Case Study for Intelligent Wireless Networks," *IEEE Network*, vol. 34, no. 3, pp. 208–215, 2020.

[16] K. Hollingsworth, K. Rouse, J. Cho, A. Harris, M. Sartipi, S. Sozer, and B. Enevoldson, "Energy Anomaly Detection with Forecasting and Deep Learning," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4921–4925.

[17] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017.

[18] W. J., "Netflix Surus GitHub," https://github.com/Netflix/Surus2015, 2015.

[19] N. Laptev, S. Amizadeh, and I. Flint, "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*. ACM, 2015, pp. 1939–1947.
[20] A. Lavin and S. Ahmad, "Evaluating Real-Time Anomaly Detection Algorithms–The Numenta Anomaly Benchmark," in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 38–44.
[21] N. Spruston, "Pyramidal neurons: dendritic structure and synaptic integration," *Nature Reviews Neuroscience*, vol. 9, no. 3, p. 206, 2008.
[22] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature neuroscience*, vol. 3, no. 9, p. 919, 2000.
[23] S. Peisert, R. Gentz, J. Boverhof, C. McParland, S. Engle, A. Elbashandy, and D. Gunter, "LBNL Open Power Data," 2017.
[24] E. Stewart, A. Liao, and C. Roberts, "Open μPMU: A real world reference distribution micro-phasor measurement unit data set for research and application development," 2016.
[25] S. Purdy, "Encoding data for HTM systems," *arXiv preprint arXiv:1602.05925*, 2016.
[26] Y. Cui, S. Ahmad, and J. Hawkins, "The HTM spatial pooler—A neocortical algorithm for online sparse distributed coding," *Frontiers in computational neuroscience*, vol. 11, p. 111, 2017.
[27] G. K. Karagiannidis and A. S. Lioumpas, "An improved approximation for the gaussian q-function," *IEEE Communications Letters*, vol. 11, no. 8, pp. 644–646, 2007.
[28] J. Hawkins, S. Ahmad, and D. Dubinsky, "Hierarchical temporal memory including HTM cortical learning algorithms," *Techical report, Numenta, Inc, Palto Alto http://www. numenta. com/htmoverview/education/HTM_CorticalLearningAlgorithms. pdf*, 2010.
[29] S. Guha, N. Mishra, G. Roy, and O. Schrijvers, "Robust random cut forest based anomaly detection on streams," in *International conference on machine learning*, 2016, pp. 2712–2721.
[30] R. P. Adams and D. J. MacKay, "Bayesian online changepoint detection," *arXiv preprint arXiv:0710.3742*, 2007.
[31] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.
[32] M. Schneider, W. Ertel, and F. Ramos, "Expected similarity estimation for large-scale batch and streaming anomaly detection," *Machine Learning*, vol. 105, no. 3, pp. 305–333, 2016.
[33] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan, "Statistical techniques for online anomaly detection in data centers," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*. IEEE, 2011, pp. 385–392.
[34] J. C. Robertson, E. W. Tallman, and C. H. Whiteman, "Forecasting using relative entropy," *Journal of Money, Credit, and Banking*, vol. 37, no. 3, pp. 383–401, 2005.
[35] E. A. Feinberg and D. Genethliou, "Load forecasting In: Applied Mathematics for Restructured Electric Power Systems": Optimization, Control, and Computational Intelligence," 2005.
[36] D. Henderson, "An advanced electronic load governor for control of micro hydroelectric generation," *IEEE Transactions on Energy Conversion*, vol. 13, no. 3, pp. 300–304, 1998.
[37] J. Wesen, V. Vermehren, and H. de Oliveira, "Adaptive Filter Design for Stock Market Prediction Using a Correlation-based Criterion," *arXiv preprint arXiv:1501.07504*, 2015.
[38] K. J. Lang, A. H. Waibel, and G. E. Hinton, "A time-delay neural network architecture for isolated word recognition," *Neural networks*, vol. 3, no. 1, pp. 23–43, 1990.
[39] Y. Cui, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *Neural computation*, vol. 28, no. 11, pp. 2474–2504, 2016.
[40] J. R. Zeidler, "Performance analysis of LMS adaptive prediction filters," *Proceedings of the IEEE*, vol. 78, no. 12, pp. 1781–1806, 1990.
[41] A. Waibel, "Consonant recognition by modular construction of large phonemic time-delay neural networks," in *Advances in neural information processing systems*, 1989, pp. 215–223.
[42] Numenta, https://github.com/numenta/htmresearch/blob/master/projects/sequence_prediction/continuous_sequence/run_lstm_suite.py, 2016.
[43] S. R. Chhetri, A. Barua, S. Faezi, F. Regazzoni, A. Canedo, and M. A. Al Faruque, "Tool of spies: Leaking your ip by altering the 3d printer compiler," *IEEE Transactions on Dependable and Secure Computing*, 2019.
[44] A. Barua and M. A. Al Faruque, "Hall Spoofing: A Non-Invasive DoS Attack on Grid-Tied Solar Inverter," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 1273–1290.

Anomadarshi Barua Anomadarshi Barua received his B.Sc. degree in Electrical and Electronic Engineering (EEE) from Bangladesh University of Engineering and Technology (BUET) in 2012. He received his M.Sc. degree in Embedded Computing System jointly from University of Southampton (UoS), UK and Norwegian University of Science and Technology (NTNU), Norway in 2016. He is currently working toward his Ph.D. at University of California, Irvine, USA. His research interests include hardware security, computer micro-architecture, smart grid, and sensor security.

Deepan Muthirayan Deepan Muthirayan received his B.Tech. degree in Engineering Design from Indian Institute of Technology, Madras, India in 2010. He received his M.Sc. and Ph.D. degree in Mechanical Engineering from University of California, Berkeley, USA in 2012 and 2016, respectively. He is currently working toward his postdoctoral degree at University of California, Irvine, USA. His research interests include algorithms for learning and decision making in networked systems, game theory, and nonlinear adaptive control.

Pramod P. Khargonekar Pramod P. Khargonekar received B. Tech. Degree in Electrical Engineering in 1977 from the Indian Institute of Technology, Bombay, India, and M.S. degree in Mathematics in 1980 and Ph.D. degree in Electrical Engineering in 1981 from the University of Florida, respectively. He has been on faculty at the University of Florida, University of Minnesota, The University of Michigan, and the University of California, Irvine. He is currently the Vice Chancellor for research and distinguished professor of Electrical Engineering and Computer Science at the University of California, Irvine. His research interests include renewable energy, smart grids, grid cybersecurity, flight control applications leveraging deep learning, adversarial systems, and neural system modeling and control.

Mohammad Abdullah Al Faruque Mohammad Abdullah Al Faruque received his B.Sc. degree in Computer Science and Engineering (CSE) from Bangladesh University of Engineering and Technology (BUET) in 2002, and M.Sc. and Ph.D. degrees in Computer Science from Aachen Technical University and Karlsruhe Institute of Technology, Germany in 2004 and 2009, respectively. Mohammad Al Faruque is currently with the University of California Irvine (UCI), where he is an associate professor (with tenure) and directing the Cyber-Physical Systems Lab. His research interests include renewable energy, smart grids, grid cybersecurity, system security, health IoT, hardware security, autonomous vehicles, and graph based neural system modeling.