Proceedings of the 2023 Winter Simulation Conference
C. G. Corlu, S. R. Hunter, H. Lam, B. S. Onggo, J. Shortle, and B. Biller, eds.

SIMULATION OF A NOVEL, LOW SWAP, SPARSE HYPER-DIMENSIONAL NEURAL NETWORK ARCHITECTURE FOR ANOMALY DETECTION AI AT THE EDGE

Dean C. Mumme
Ksenia Burova

RAM Laboratories, Inc.
591 Camino de la Reina
San Diego, CA 92108, USA

ABSTRACT

This paper details the simulation and performance results of a Sparse Hyper-Distributed Robust Efficient Neural Network (SpHyRE-Net) architecture that performs anomaly detection for real-world time-series data. SpHyRE-Net is an innovative, novel, low size, weight and power (SWaP) machine learning solution for devices operating at the tactical edge. It utilizes bit operations and sparse hyper-dimensional representations for bio-inspired learning via a Hebbian-like rule that results in a combined power-latency reduction of 2-orders of magnitude over ordinary deep networks. The paper details the application of SpHyRE-Net to real-world cell-traffic datasets as well as the simulation requirements needed to minimize latency and memory use. Also discussed are the mechanisms necessary for implementing the architecture on an FPGA as a precursor to realization on a neuro-morphic ASIC with ultra-low power profile.

1 INTRODUCTION

Here we detail the development of bio-inspired neural-network architecture called Sparse Hyper-Distributed Robust Efficient Neural Networks (SpHyRE-Net) to provide a 3rd generation AI capability for anomaly detection over time-series data on small size weight and power (SWaP) edge devices. The primary goals included simpler digital logic operations compared with the multiply-accumulate cycles (MAC) of deep neural networks (DNN) in order to greatly reduce latency and power consumption. The ultimate objective for future work is the implementation of a high-performing, very low-power neuro-morphic application-specific integrated circuit (ASIC), capable of performing military relevant communication and edge sensing application(s). These gains are expected in comparison to state-of-the-art MAC-based DNN such as long short-term memory (LSTM) and recurrent neural networks (RNN). This paper reports our accomplishments in developing a software simulation prototype for SpHyRE-Net that greatly reduces the compute burden compared to conventional DNNs.

Figure 1 shows the high-level concept of the SpHyRE-Net architecture that has been implemented as a multi-threaded simulation. A dataset representing time-series data encoded and presented to the input of a spatial-pooling layer consisting of columns of cells (proximal layer) with each column acting as a single perceptron that counts the number of active inputs and becomes active if the count is above a threshold. Cells within the column (distal layer) form a temporal layer in which cells within a column attempt to predict whether the column will become activated by the next input from the time-series. A count of which columns were correctly predicted is presented to a statistical threshold unit to determine whether this number has deviated from normative, given the past activity of this indicator. When the number of correct

predictions becomes too low, an anomaly is flagged for the time-series at this point. The objective of the system is to adjust weights within the proximal and distal layers over time to learn the normative characteristics of the time-series in order to detect when a significant deviation occurs. More details of the system are described below.

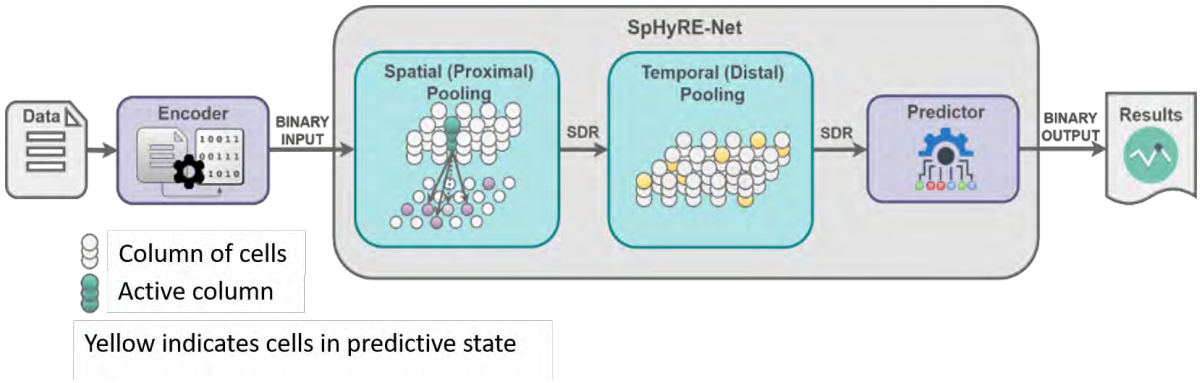

Figure 1: Overall SpHyRE-Net architecture to be simulated.

The SpHyRE-Net simulator was inspired by hierarchical temporal memory (HTM) theory called the "Thousand Brains Theory of Intelligence" (Baralacchi 2015; Hawkins 2016) to model biological learning in the brain. As an alternative state-of-the-art learning algorithm, the HTM is based on a theory of human neocortex, and has shown results in performing anomaly detection Hirakawa (2021). The HTM algorithm substantially differs from traditional DNNs by employing sparse hyper-dimensional vectors (post-encoding dimension > 500) in order to enable training and query cycles using massively parallel bit operations. The model performs sequence learning over a time-series of such vector embeddings in order to model the series and detect anomalies according to the current model. This allows the model to continually update itself to handle non-stationary sequences.

Besides demonstrating the effectiveness of SpHyRE-Net for detection of anomalies in time-series data, the primary objective was to determine whether the simulator could be architected in a way that facilitates the eventual realization as a Field-programmable gate array (FPGA) as a prelude to realization as an application specific integrated circuit (ASIC). In this paper, we describe standard HTM-based architecture and operation, the innovations introduced to SpHyRE-Net by this investigation for the implementation of the simulator and eventual realization as an FPGA. We then discuss the datasets for evaluating the model, and finally our experiments, results and conclusions.

2 HTM-BASED ARCHITECTURE AND OPERATION

2.1 Original HTM Architecture

The model is defined in terms of the input dimension, N, the number of columns (also N), each containing M cells. The model is fed by a (filtered) time-series data-stream and each value arriving from the stream is encoded as a sparse distributed representation (SDR) onto a N-dimensional binary "data vector", D_i ,for $i = 0,1,2,\ldots N$ and presented to the model's input layer at the beginning of its "input cycle" (Figure 2).

Every column is a perceptron fed by a "proximal segment" of binary links that samples a sparse subset of the input units. A link is considered to be "potential" or "connected" as determined by the connection's "permanence" parameter having a value between 0 and 1. If the **permanence** value is above a prespecified threshold, T_c, then the connection is considered to be "connected" with a weight of 1, otherwise it is called "potential" and has a connection weight of 0. During training the permanences are adjusted using a Hebbian rule, and this causes the binary links to switch between being connected and partial. However only activated columns are allowed to adjust their permanences. A column becomes active whenever the number of active inputs on its connected links exceeds a threshold, activation threshold T_a.

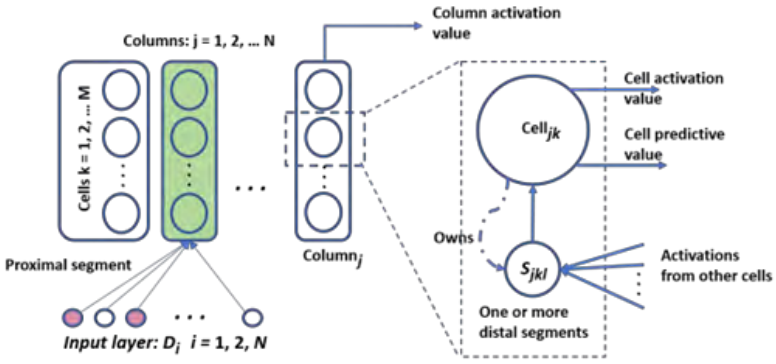

Figure 2: Standard HTM Architecture.

Each of a column's cells "owns" one or more perceptrons, called "distal segments", that sample the activations of a sparse random selection of the other network cells. The activation state of a particular segment is only provided to the owning cell. The rule for adjusting permanences is the same for proximal and distal segments.

The HTM has a two-phase "input cycle" (or "cycle"), each cycle is identified by a sequential integral time-value, t, starting at 0 for the first cycle. The first phase starts with the excitation of the proximal layer with the input vector to activate the columns. The second phase starts with the excitation of the distal layer in order to make predictions for the next cycle. Between the first and second phases a score is calculated over the predictions that were made during the end of the second phase of the prior cycle.

The goal of the network for each cycle is to predict which of the columns will become activated in the following cycle. Predictions are made at the end of the cycle by the distal segments, each monitoring a sparse sample of the cell-activation space in order determine when to place its owning cell into the "predictive state".

To provide input to the distal segments, each column activated in the first phase will activate any of its cells that were placed in the predictive state by the previous cycle and also declare them as "winners". If no such cell exists, the column was not predicted and so it "bursts" by activating all its cells. In such case it must select one of them as the "winner". All winner cells will subsequently be made to learn by growing a new distal segment or by adjusting permanences on an existing one as will be described.

During the second part of the cycle, all cell activations are forwarded to the distal segments and a sparse subset of these segments become active. Whenever a cell has at least one activated distal segment, it goes into the "predictive state" (becoming a "predictive cell") meaning that it is predicting that its column will become active during the next input cycle. During the first phase of the subsequent cycle, an active column having one or more cells in the "predictive state" is correctly predicted. Likewise, an inactive column having no cells in a predictive state is correctly predicted. An error-score is generated by counting the columns that are not correctly predicted.

2.2 HTM Training

The system learns in several ways as follows: 1) The proximal connections between the inputs and the columns learn by a Hebbian rule, but only when a column is active. 2) When a distal segment becomes activated, each of its permanences is strengthened when connected to an active cell and weakened otherwise; 3) When a distal segment "mis-predicts" by placing the owning cell into the predictive state and the corresponding column does not activate in the next cycle, the segment is "punished" by decaying all its permanences by some small factor; 4) When a winner cell is selected having the fewest distal segments, it grows a new segment that samples the currently active cells; 5) When a new set of winner cells is determined for the current cycle, a segment from each such cell is selected to grow new synapses from other active / winner cells.

Over time some columns can become chronically inactive and never learn. A mechanism used to prevent this is called "boosting". The mechanism tracks the activity of columns over time and for those having significantly less activity than average will have their activation count boosted by some factor in order to increase the probability they will activate.

The model has many meta-parameters, including: The learning rate or "permanence delta", one value for augmentation and another for diminution; the thresholds, T_c and T_a; the fraction for diminution of permanences in punished segments; and the boost strength. For the full parameter list, see Barlacchi (2015).

3 SPHYRE-NET SIMULATOR AND ITS INNOVATIONS

3.1 Simulation Mechanisms

To evaluate its performance, SpHyRE-Net was implemented as a multi-threaded simulator in C++ and executed on a Linux operating system. The host was a Core™ i7-7700HQ based machine running at 2.8GHz with 4 multi-threaded cores and 32GB of memory. In order to facilitate compatibility with eventual implementation in an FPGA and to account for the recurrent signal flow, the simulator was implemented as an activation-loop of perceptrons as shown in Figure 3.

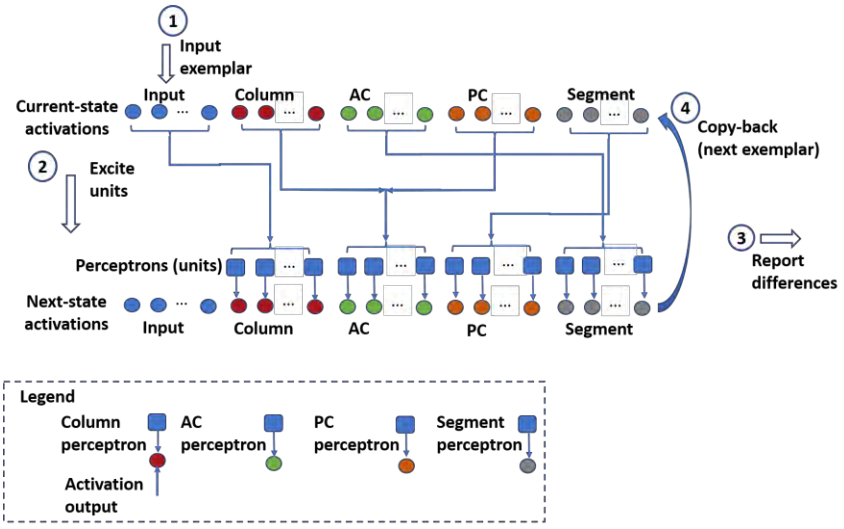

Figure 3: SpHyRE-Net simulator's activation loop to support recurrent activations.

The collective simulation state is composed of two vectors of "activation values", the current state vector, and the next state vector. The activations vectors are a concatenation of the input activations, "column" activations, activations of the "AC-units" and the "PC-units". These and the concept of "segments" as be described in the sequel.

Operation is as follows: 1) Initialize the current-state with the activations of the input units as inputs to the column units, 2) Project the column units resulting activation values to the next-state vector, 3) After the second iteration of the loop, report changes in activations, 4) Replace the activations in the current-state vector with those of the next-state vector, and repeat—allowing for new values of the input activations at step 1). The key performance gain entailed using multiple threads, each computing a dot-product rule using the logical ANDing of bits within bit-vectors representing the "on-off" state of the column units and the cells.

To control loop synchronization, we utilized the C++11 synchronization library. Multiple threads handled the computation of the activation levels to be propagated to the next-state vector. An atomic counter was used for tracking how many of the threads had completed the task, so that steps 3) and 4) would be started (using a single update thread) only after all threads were finished. An atomic flag was used to indicate the completion of 4) in the update thread so that the activation threads could begin the next loop.

Finally, since the algorithm requires non-determinism for various operations, such as for selecting "winners", a concurrent random number generator, `rand_r` (GCC linux) was utilized for thread-safety.

3.2 SpHyRE-Net Innovations for Realizing Simulator in Hardware

To diminish latency, enhance performance and make the simulated architecture suitable to hardware implementation, the innovations were included in the design:

- No segment punishment: In contrast to the original HTM, segments in SpHyRE-Net that mis-predict their corresponding column do not have their weights reduced.
- Soft winner-take-all (SWTA): SWTA is performed over the columns and the distal segments in order to stabilize the response of the network.
- Dot-product activation: Instead of binary activation values, a linear activation is used to produce a continuous valued output activation value.
- Bounded segment growth: The growth of new segments and of new connections for each segment are both limited to relatively small numbers (5 or 10).
- Split modality for excited cells: The cells within a column are each split into two units, a "predictive cell" (PC) and an "activation cell" (AC).
- Statistical analysis of prediction outputs: The statistical likelihood in the number of false positives and false negatives per cycle is evaluated over time to get an anomaly score.

Importantly, an architectural modification of the "cell units" was made in order to facilitate potential realization in FPGA hardware. This required implementing all neuronal components into "perceptron-canonical form," so that the functionality could be obtained using simple perceptron-based neuronal units. For this, each of the cells within a column was split into two "units", namely an "activation cell" (AC unit) and a "predictive cell" (PC unit) with the AC units being activated during the first phase of a cycle and the PC units being activated during the second (Figure 4). The activation of the AC units represents ordinary cell activation fed to the distal segments and PC-unit activation represents the cell's "predictive state."

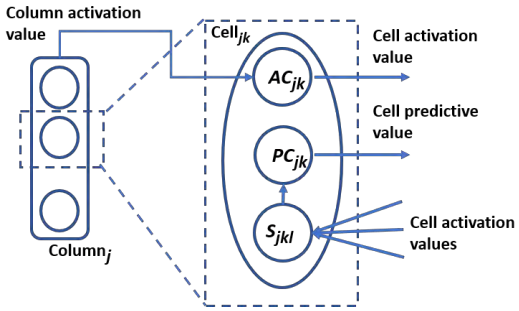

Figure 4: Realization of cell-units as activation cells and predictive cells.

Note that each column owns its cells with each cell composed of an AC and a PC unit and owning one or more segments, S. These entities, including the segments can be considered as being part of the cell's column. Particularly, the distal segments can be grouped according to the columns they serve, each segment providing a signal as to whether it predicts that its column will become active in the next cycle.

The signal flow for the network is shown in Figure 5. The encoded input data vector, D feeds into each of the columns, C, over via proximal segments, one segment per column. A column influences each of its cells AC_{jk} via its activation state and the "winner signal". The AC units provide their activation values to the distal segments, S, with each segment providing input to a single PC unit that is part of the segment's owning cell. Every PC has at least one such segment, and can grow new ones as the system learns. Whenever at least one of the PCs segments becomes active, the PC activates to indicate that the cell is in the predictive state. The dashed line on the bottom left of the figure indicates that the predictive signal is

computed during the second phase of the previous cycle, and is provided to all AC units in the same column during the first phase of the current cycle.

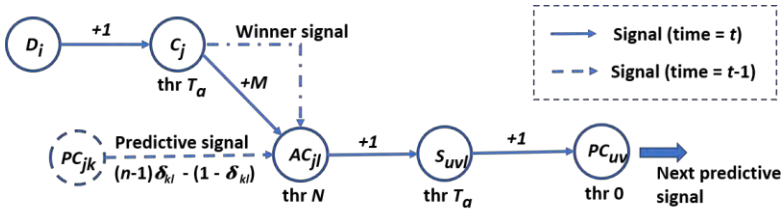

Figure 5: Activation and prediction signal flows.

Figure 6 illustrates computation of the error score for a given input vector. For the current input cycle, at time-step, *t*, a vector of activation values for all columns can be aligned with a similar vector of prediction values, generated at the end of the previous input cycle (time-step *t*-1). The number of false positives (FP) and false negatives (FN) are computed by element-wise comparison and number of errors. To obtain a "raw-anomaly score" (RAS) the number of errors is divided by the number of active columns. The statistical behavior of the raw anomaly score over time is used to determine whether the model will flag an anomaly.

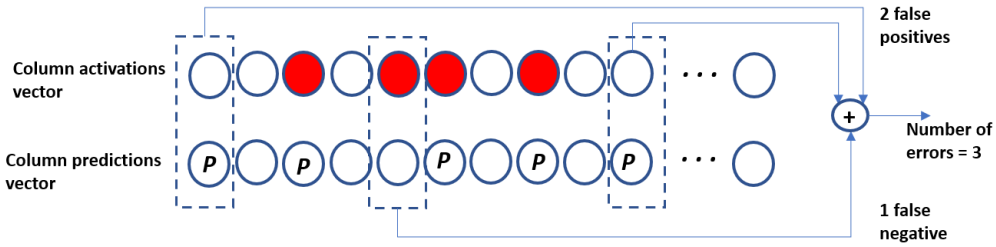

Figure 6: Computation of raw anomaly score.

To determine whether an anomaly has occurred in the time-series data we compute mean, μ, and standard-deviation, σ, of the RAS over the past (say 100) time-steps. Given a threshold number, *T*, set the "anomaly flag" to true if the score, *s*, is more than *T* standard-deviations above the mean. Thus, we have

$$\texttt{Anomaly-flag} = s \geq \mu + T * \sigma,$$

where the value of *T* was set to 2.5 for all experiments.

Whereas the various network segments have permanences for plasticity, the AC and PC units are "hard wired" as shown by the weight-values on the links shown in Figure 5. This is done in order to realize the intended functionality as a network of interconnected perceptrons. The intended behavior of the AC and PC units is summarized as follows:

- Cells of an inactive column do not come on.
- When a column is activated, it activates all of its cells that are already in a predictive state.
- When no such cell exists for the column, all cells are activated (column bursts).
- An active distal segment will trigger its owning cell to go into a predictive state.

An activated column that has been predicted by at least one PC will activate only the corresponding AC unit(s). So we give the link from the column to each of its AC units the value of *M*, and the AC itself is given a threshold of *M* where *M* is the number of cells per column. The AC will weigh the corresponding

PC unit with a weight of $M - 1$. It will weigh all other PCs for the column with a weight of -1. This is indicated at the lower left of Figure 5. The connecting weight from PCj into ACi of the same column is

$$w_{ji} = (M - 1)\delta_{ji} - (1 - \delta_{ji}),$$

where δ_{ij} is the Kronecker-delta. This expression evaluates to $M - 1$ for $i = j$ and -1 otherwise.

This means that a PC strongly excites its corresponding AC unit and inhibits the others. When the column activates, all its ACs will activate if no PC is active (the column was unpredicted and so "bursts"). If instead, at least one PC is on, AC's not having an active PC will be inhibited enough not to reach their activation threshold, and so only the ACs with active PCs will become activated. The weights as stated therefore accomplish the intended behavior.

For the case that the column bursts, a winner is declared which labels its PC unit as a candidate for training. If instead an active column is not bursting, the winner flag is attached to all the column's predictive PCs and they are subsequently trained.

3.3 Simulation Efficiency

The simulations were performed on an Intel core-I7, 7th generation PC with 4 hyper-threaded processors, or 8 "logical processors". Due to the large number of connections within the simulated neural network, various data-structures and update strategies were used to eliminate potentially redundant computation. A major gain against latency can be realized by placing an activated unit (column or segment) on a list of active units so that subsequent permanence updates are only performed over the lists. For this, the columns are divided amongst the threads and each thread updates its own list. A synchronization barrier is used to determine when all threads have completed and then a single thread merges the lists and selects winner units. The lists are then subdivided over multiple threads which then update the permanence values of the column units.

Similarly, the fact that only active segments have their permanences adjusted allows multiple threads to collect segments as they activate so that a single thread can combine the results and redistribute the activated segments to threads that update the permanences. A synchronization barrier then allows a single thread to collect the prediction statistics and indicate an anomaly score.

A key parameter for making this work is the activation "sparseness" that is imposed on the columns and segments. The sparseness is controlled by the activation thresholds which allow a small number of units at each of the column and segment layers in order to reduce the lists to a manageable size, but must also allow enough units to come on so that when they are sparsely sampled by connections into the next layer, there will be sufficient activity to activate units in the succeeding layer. Experience confirms that a activation of a few dozen units (40 to 50) per layer is sufficient.

4 DATA PREPARATION AND ENCODING

4.1 The Dataset

The experiments utilized a publicly available dataset produced by the Telecom Italia Mobile corporation and used by the Big-data Challenge competition Barlacchi (2015). The dataset is the result of a computation over the Call Detail Records (CDRs) generated by the Telecom Italia cellular network over the city of Milano, Italy. The CDRs log the user activity for billing purposes and network management.

The dataset is a collection of cellular activity records collected by dividing the city into a rectangular grid, and traffic-rates in each cell of the grid was recorded. The cells were numbered from 1, 2, ... onward. For the initial experiments, we used data from the first 3 cells to study anomaly detection performance.

The data utilized was a post-processed version proposed by Subutai (2017) that consists of cellular activity records from 11/01/2013 through 12/30/2013 for the morning, afternoon and evening activity and divided into 18 temporal intervals per each portion of the day.

Each record field details the statistics for a particular communications cell as shown here:

- `Timestamp`: the calendar date
- `Time slot`: temporal interval during which the data was collected
- `SMSin`: the number of received SMS messages
- `SMSout`: the number of sent SMS messages
- `Callin`: the number of inbound calls
- `Callout`: the number of outbound calls
- `Internet`: the amount of internet access generated

Derived datasets having one or more of the above features were constructed to train both single feature and multi-feature data.

4.2 Low-pass Filtering

To enhance simulator performance, we applied a low-pass filter (LPF) to the data-stream before encoding. The LPF was implemented in terms of a sequential state, s_t , at time t:

$$s_0 = \texttt{input}(t=0);$$

$$s_t = \alpha \cdot \texttt{input}(t) + (1-\alpha) \cdot s_{t-1}, \quad t \geq 1,$$

where α is a fixed parameter between 0 and 1. Initial work varied the value of α over the values 0.1, 0.05 and 0.033. The value α = 0.05 was determined to be suitably performant for all subsequent experiments in that it reduced noise significantly without sacrificing overall signal detail.

Figure 7 shows the effects of low-pass filtering (LPF) on the cell-3 `internet` traffic. The original dataset consists of 3262 exemplars and is shown in the first trace. The time axis is shown as "INDEX" representing the time-value, *t*, and the vertical axis represents the number of existing network connections for a particular 10-minute interval in Milan. The trace has numerous very tall peaks that don't reflect the general nature of the data. The second trace shows the data after it was filtered with $\alpha = 0.05$.

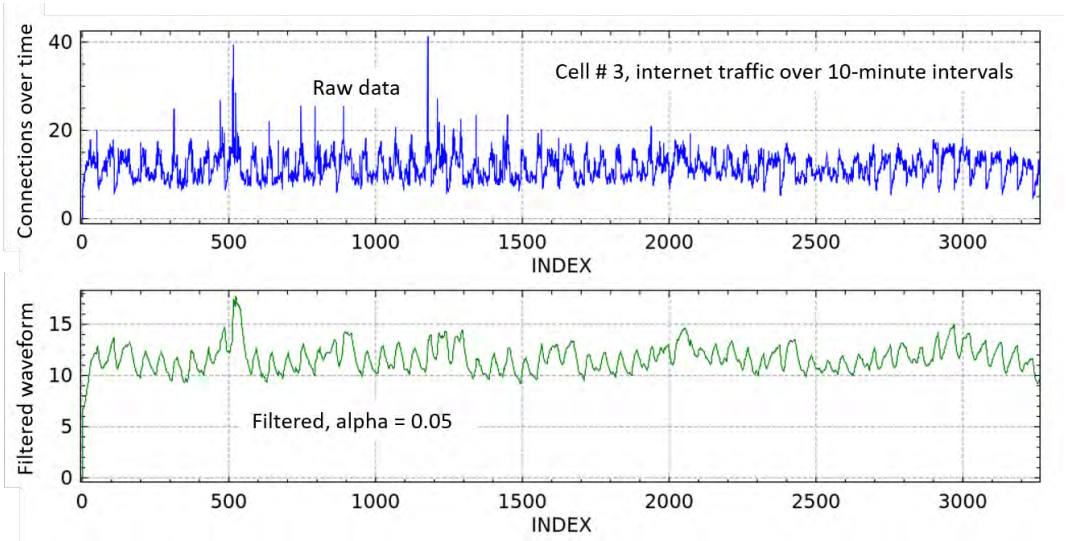

Figure 7: Internet traffic data before and after the use of a low-pass filter.

4.3 Encoding

We based our encoding approach on the random distributed scalar encoder (RDSE) Zhu (2013) which uses a fixed total number of 1-bits distributed randomly over the *N*-dimensional input vector. The encoder determines an encoding for a baseline data-value, and then changes the bit-pattern slightly as the data-value is gradually changed. The result is an encoding that preserves the semantics of distance in that similar data-values result in similar bit vectors as measured by hamming distance.

The data values are discretized into buckets and adjacent buckets have encodings that overlap strongly. We found that SpHyRE-Net performs best when not more than about 135 buckets are generated. This determined the granularity of discretization which is computed as the range of the feature values divided by this number.

Encoding API within the SpHyRE-Net simulation allowed us to stack encodings for multiple feature values in order to construct a uniform SDR that gets fed into the SpHyRE-Net algorithm. Each channel can be turned on/off and two types of encoding can be chosen for numerical values, RDSE and simple scalar encodings. There were 6 features total encoded in our experiments – `time-slot`, `smsin`, `smsout`, `callin`, `callout`, and `internet`. All features were encoded with RDSE except for `time-slot`. For that we used simple scalar encoding which assigns bits to the vector-embedding according to a sliding window having its position determined by the value encoded. The window positions for successive values of the field have high bit-wise similarity to preserve distance semantics.

Experiments utilized networks having of N = 256, 512, and 2048 where N is both the input dimension and the number of network columns. Initial experiments tested encodings having from 5% to 15% active bits. Networks of size 512 and larger could remain performant as the number of active bits is decreased for greater sparseness. For the larger networks, the value 5% provided the best overall results considering that the computational burden is greatly reduced as sparseness increases. When encoding multiple features, the embedding was partitioned by feature, and we provided each feature with the same or similar size bit-field. For the time slot feature, we always provided 21 total bits for all sizes of encoded SDR.

5 RESULTS

In our experiments, the SpHyRE-Net simulator was run over datasets of 3262 exemplars each. Single-feature datasets were each constructed from one of: `smsin`, `smsout`, `callin`, `callout` and `internet`. Multi-feature datasets were generally constructed from two selected features, although most multi-feature testing was done using a dataset incorporating all these features encoded into each exemplar. We also experimented using `time-slot` combined with another feature such as `internet` to determine whether SpHyRE-Net's performance would benefit from temporal information.

For all feature types, the data generally follows a weekly pattern of peaks and troughs giving a quasi-periodic waveform with a period of one day. The shape of the waveform and height of the peaks tend to follow one pattern for weekdays and another for the weekends making it easy for a human to judge both the time-of-week and the time-of-day over the course of the data stream. This makes it ideal for testing whether SpHyRE-Net can learn the overall pattern and flag novel trends as they occur. Our experiments have shown that during the first 500 or so exemplars, SpHyRE-Net emits high anomaly scores and then gradually diminishes them as it learns the pattern. At that point it becomes more selective when flagging an anomaly.

5.1 Anomaly Detection

Figure 8 shows results for running SpHyRE-Net over filtered data representing internet traffic on Milan-cell 3. The bottom waveform is the filtered input data, the upper waveform represents the raw error-scores provided by SpHyRE-Net, and the vertical bars over the LPF data indicate the presence of an anomaly according to the statistical scores.

We see peaks in the raw scores and the corresponding vertical bars at the same locations. The most notable anomaly is the tall peak in the filtered data at exemplars numbered between about 500-550 indicating a sudden burst in traffic. The next two anomalies correspond to unlikely maxima in the

waveform, and finally, between exemplars 2900 and 3000 two anomalies indicate an irregularity in the overall pattern in which the traffic follows a descending trend over the following week, beginning with abnormally high traffic for Wednesday of the last whole week.

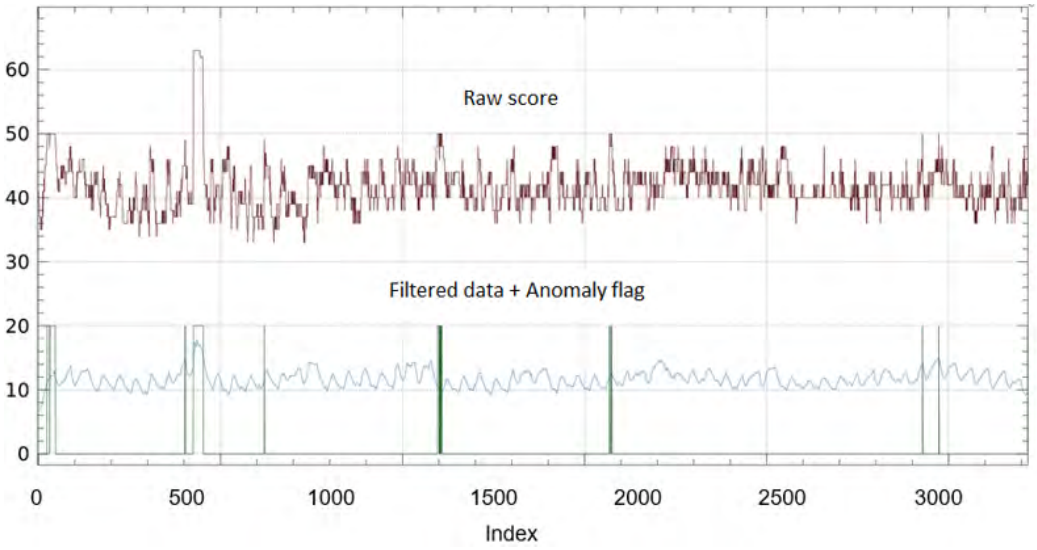

Figure 8: Filtered Data, raw scores, and anomaly flags over a set of 3262-point time-series.

Another example is shown in Figure 9. The upper trace shows filtered `callin` traffic for cell 4. The second chart represents the raw error scores, and the bottom chart shows the anomaly indicator bars. In this trace the weekday traffic is more voluminous than that of the weekends.

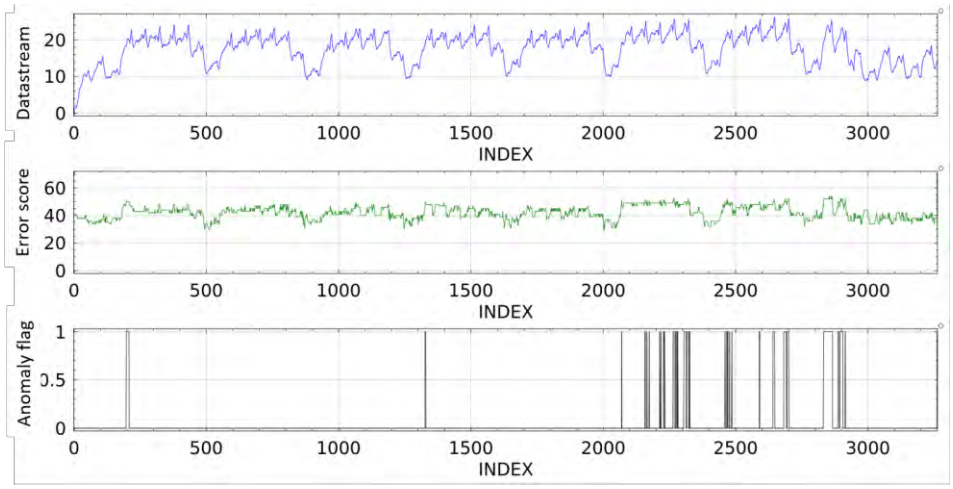

Figure 9: SpHyRE-Net analysis of filtered cell 4 `callin` data.

The most notable response of SpHyRE-Net begins approximately at exemplar 2050 and extends to the right. For this portion of the data, the weekday traffic is larger than for previous weeks. This is indicated by the high 'plateau' in the raw error scores starting at about the same location. The condition continues for 2 weeks, although SpHyRE-Net responds less to the anomalies of the second week, likely due to the fact that learning is turned on throughout the time-series. This means that SpHyRE-Net has begun learning the trend for the first of these two weeks is normative. Since the 2nd week is similar to the first, SpHyRE-Net is less likely to indicate the presence of anomalies.

For the "final week" in the data series, the first two days have abnormally high traffic and then the week's traffic reduces markedly. The waveform for this week is very different from the previous weeks. SpHyRE-Net flags this anomaly at the beginning of the week but does not respond to the rest of the week.

We believe that this is due to the fact that the occurrence of a large anomaly tends to remove the 'context' within SpHyRE-Net's inner state. This will be reset if the dataset returns to its former statistical form in succeeding weeks. Otherwise, SpHyRE-Net will learn the new statistical pattern as being normative. This represents adaptation to non-stationary data.

5.2 Simulator Performance

We found that learning and accuracy are affected by various factors including:

- Learning 'picks-up on' cycles within the data without the use of the timestamps, allowing anomaly detection over segments within quasi-periodic data.
- Low-pass filtering of data before encoding greatly enhances performance on noisy datasets.
- Statistical analysis of the raw error-scores makes anomaly detection more precise.

As described in the innovations section we made major alterations to the baseline HTM algorithm, primarily for performance enhancement. First, the baseline HTM performs punishment of distal segments that predict a column that does not subsequently activate. However, we obtained acceptable performance without segment punishment and enhance the number of predictions made for each cycle. This in turn increased prediction accuracy. It also led to nearly a 50% speedup in processing each exemplar since the algorithm did not have to update all connections of all mis-predicting segments for each cycle.

Whereas the standard HTM employs soft winner-take-all (SWTA) to regulate the number of columns that come on for each cycle, we found that SWTA was also useful for similarly regulating the distal segments. This stabilized the number of predictions made at each cycle. It also allowed bringing the number of predictions up nearer to the number of columns that activate during the cycle.

We eliminated the threshold at the output of all segment units and utilized the permanences as connection weights for a dot-product activation rule to get continuous values at the segment output. This significantly enhanced anomaly detection.

Finally, we had to limit distal segment growth, both in total number of segments allowed to grow per cell, and the total number of new connections that can be added to a segment. Without bounds, a huge penalty is encountered both in time and in space. This is why most experiments used an input dimension of $N = 512$, with the same number of columns and $M = 16$ or 32 cells per column. Our experiments showed that changes in the number of cells per column little or no difference in performance. Likewise, as long as the operational suggestions of Barlacchi (2015) were followed, few meta-parameters had much impact on performance. Our method of calculating the statistical likelihood over the time-averaged error-score is that of Hiralawa (2021).

The following is a summary of our observations of SpHyRE-Net anomaly detection performance:

- Networks having at least 256 columns performed similarly. Larger networks allowed for more sparseness as long as downstream segments sampled enough of the activation space.
- Larger networks with a corresponding larger input space had a better capacity for encoding multiple features into the sparse vector embeddings thereby improving performance over multi-feature data.
- As stated earlier, segment punishment did not enhance performance but cost a lot of computation.
- The ordinary HTM utilizes SWTA over the columns to force the same number to come on each cycle. We found that applying SWTA to the distal segments as well forced more cells to be placed into the predictive state and enhanced performance.
- For our experiments, encoding a timestamp into the embeddings hurt anomaly detection performance, so they were not included in our final runs.
- The algorithm is somewhat tolerant to changes in the number of cells per column and changes in meta-parameter values.

5.3 Conclusions and Future Research

In addition to providing a means for learning to detect anomalies in time-series data, this effort demonstrated a means for realizing the SpHyRE-Net simulator in terms of interconnected perceptron units in order to facilitate realization as an FPGA.

For detection of anomalies within the cellular datasets, datasets for each traffic type appeared to establish a pattern over the first few weeks, and then deviate from it toward the end. This seemed apparent to the human eye and was generally flagged by SpHyRE-Net. However, it is not currently known how SpHyRE-Net "models the data" and so further tests over other dataset-types conducted in order to determine the degree to which SpHyRE-Net can serve in critical systems applications.

Finally, for larger input dimensions of 1000 or larger, the algorithm imposes massive memory requirements, especially if segment growth is not sharply limited. Future investigations can be made to determine how these considerations can inform effective FPGA and ASIC designs.

ACKNOWLEDGEMENTS

This work was funded from DARPA's Hyper-Dimensional Data Enabled Neural Networks (HyDDENN) program (Agreement #HR00112090040, PR # HR0011045180).

REFERENCES

Barlacchi, G., M. Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland, and B. Lepri. 2015. "A Multi-Source Dataset of Urban Life in the City of Milan and the Province of Trentino". *Scientific Data* 2(1):1-15.

Hawkins, J. et al. 2016. "Biological and Machine Intelligence". Release 0.4. https://numenta.com/resources/biological-and-machine-intelligence/, assessed 15^{th} June, 2020.

Hirakawa, R., H. Uchida, A. Nakano, K. Tominaga, and Y. Nakatoh. 2021. "Anomaly Detection on Software Log Based on Temporal Memory". *Computers & Electrical Engineering* 95:107433

Hole, K.J., and S. Ahmad. 2021. "A Thousand Brains: Toward Biologically Constrained AI". *SN Applied Sciences* 3:743.

Kanerva, P. 2009. "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors". *Cognitive Computing* 1:139–159.

Subutai, A., A. Lavin, S. Purdy, and Z. Agha. 2017. "Unsupervised Real-Time Anomaly Detection for Streaming Data". *Neurocomputing* (262):134-147.

Zhu, Y., L. Ni. and B. Li. 2013. "Exploiting Mobility Patterns for Inter-Technology Handover in Mobile Environments". *Computer Communications* 36 (2):203–210.

AUTHOR BIOGRAPHIES

DEAN C. MUMME is a Senior Staff Scientist at RAM Laboratories who specializes in cyber-security, simulation, and artificial intelligence research. He has performed research for numerous agencies including DHS, OSD, ONR, NAVWAR, AFRL, MDA, and DARPA. Security innovations include creation of out-of-band intrusion detection and prevention systems. His machine learning research includes ML-based detection time-series anomalies and automated correction of software errors. Patents include a novel machine-learning algorithm, a privacy preserving crowd-source event collection algorithm for highway traffic monitoring, and an out-of-band hiding of control-flow and data for cyber security. His education includes a B.S. in Aeronautics and Astronautics from the Massachusetts Institute of Technology, M.S. in Mathematics at Idaho State University, and Ph.D. in Computer Science from the University of Illinois, at Champaign, Urbana. His e-mail address is dmumme@ramlabs.com.

KSENIA BUROVA is a Senior Member of the Technical Staff at RAM Laboratories where she is working on machine learning applications in the areas of anomaly detection, situational awareness, cyber security, and the Internet of Things (IoT). Previously, Ms. Burova interned at Oak Ridge National Laboratory where she performed software engineering on DOE's flagship energy modeling simulation engine, EnergyPlus. She has both a B.S. and a M.S. in Computer Science from the University of Tennessee. Her e-mail address is kburova@ramlabs.com.