Performance Analysis of Sparse Matrix Representation in Hierarchical Temporal Memory for Sequence Modeling

Csongor Pilinszki-Nagy[1] and Bálint Gyires-Tóth[2]

***Abstract*—Hierarchical Temporal Memory (HTM) is a special type of artificial neural network (ANN), that differs from the widely used approaches. It is suited to efficiently model sequential data (including time series). The network implements a variable order sequence memory, it is trained by Hebbian learning and all of the network's activations are binary and sparse. The network consists of four separable units. First, the encoder layer translates the numerical input into sparse binary vectors. The Spatial Pooler performs normalization and models the spatial features of the encoded input. The Temporal Memory is responsible for learning the Spatial Pooler's normalized output sequence. Finally, the decoder takes the Temporal Memory's outputs and translates it to the target. The connections in the network are also sparse, which requires prudent design and implementation. In this paper a sparse matrix implementation is elaborated, it is compared to the dense implementation. Furthermore, the HTM's performance is evaluated in terms of accuracy, speed and memory complexity and compared to the deep neural network-based LSTM (Long Short-Term Memory).**

***Index Terms*—neural network, Hierarchical Temporal Memory, time series analysis, artificial intelligence, explainable AI, performance optimization**

I. INTRODUCTION

Nowadays, data-driven artificial intelligence is the source of better and more flexible solutions for complex tasks compared to expert systems. Deep learning is one of the most focused research area, which utilizes artificial neural networks. The complexity and capability of these networks are increasing rapidly. However, these networks are still 'just' black (or at the best grey) box approximators for nonlinear processes.

Artificial neural networks are loosely inspired by neurons and there are fundamental differences [1], that should be implemented to achieve Artificial General Intelligence (AGI), according to Numenta [2], [3]. [3] They are certain that AGI can only be achieved by mimicking the neocortex and implementing those fundamental differences in a new neural network model.

Artificial neural networks require massive amount of computational performance to train the models through many putational performance to train the models through many epochs. Also, the result of a neural network training is not, or only partly understandable, it remains a black (or at best a grey) box system. There is a need to produce explainable AI solutions, that can be understood. Understanding and modeling the human brain should deliver a better understanding of the decisions of the neural networks.

Sequence learning is a domain of machine learning that aims to learn sequential and temporal data, and time series. Through the years there were several approaches to solve sequence learning. The state of the art deep learning solutions use one-dimensional convolutional neural networks [4], recurrent neural networks with LSTM type cells [5], [6] and dense layers with attention [7]. Despite the improvements over other solutions these algorithms still lack some of the preferable properties, that would make them ideal for sequence learning [1]. The HTM network utilizes a different approach.

Since the HTM network is sparse by nature, it is desirable to implement it in such a way that exploits the sparse structure. Since other neural networks work using optimized matrix implementations, a sparse matrix version is a viable solution to that. This porting should be a two-step process: first a matrix implementation of the HTM network, then a transition to sparse variables inside the network. These ideas are partially present in other experiments, still, this approach remains a unique way of executing HTM training steps. Our goal is to realize and evaluate an end-to-end sparse solution of the HTM network, which utilizes optimized (in terms of memory and speed) sparse matrix operations.

The contributions of this paper are the following:

- Collection of present HTM solutions and their specifics
- Proposed matrix solution for the HTM network
- Proposed sparse matrix solution for the HTM network
- Evaluation of training times for every part of the HTM network
- Evaluation of training times compared to LSTM network
- Evaluation of training and testing accuracy compared to LSTM network

II. BACKGROUND

There have been a number of works on different sequence learning methods (e.g., Hidden Markov Models [8], Autore-

[1] Balatonfüred Student Research Group

[1,2]Department of Telecommunications and Media Informatics, Budapest University of Technology and Economics Budapest, Hungary

[3]Numenta is a nonprofit research group dedicated to developing the Hierarchical Temporal Memory.

E-mail: csongor.pilinszkinagy@gmail.com; toth.b@tmit.bme.hu

DOI: 10.36244/ICJ.2020.2.6

gressive Integrated Moving Average (ARIMA) [9]), however, in this paper artificial neural network-based solutions are investigated.

A. Deep learning-based sequence modeling

Artificial neural networks evolved in the last decades and were popularized again in the last years, thanks to the advances in accelerated computing, novel scientific methods and the vast amount of data. The premise of these models is the same: build a network using artificial neurons and weights, that are the nodes in layers and weights connecting them, correspondingly. Make predictions using the weights of the network, and backpropagate the error to optimize weight values based on a loss function. This iterative method can achieve outstanding results [10], [11].

Convolutional neural networks (CNN) utilize the spatial features of the input. This has great use for sequences, since it is able to find temporal relations between timesteps. This type of network works efficiently by using small kernels to execute convolutions on sequence values. The kernels combined with pooling and regularization layers proved to be a powerful way to extract information layer by layer from sequences [12], [4].

Recurrent neural networks (RNN) use previous hidden states and outputs besides the actual input for making predictions. Baseline RNNs are able only to learn shorter sequences. The Long Short-Term Memory (LSTM) cell can store and retrieve the so called inner state and thus, it is able to model longer sequences [13]. Advances in RNNs, including hierarchical learning and attention mechanism, can deliver near state-of-the-art results [14], [15], [16]. An example of advanced solutions using LSTMs is the Hierarchical Attention Network (HAN) [17]. This type of network contains multiple layers of LSTM cells, which model the data on different scopes, and attention layers, which highlight the important parts of the representations.

Attention mechanism-based Transformer models achieved state-of-the-art results in many application scenarios [7]. However, to outperform CNNs and RNNs, a massive amount of data and tremendous computational performance are required.

B. Hierarchical Temporal Memory

Hierarchical Temporal Memory (HTM) is a unique approach to artificial intelligence that is inspired from the neuroscience of the neocortex [1]. The neocortex is responsible for human intelligent behavior. The structure of the neocortex is homogeneous and has a hierarchical structure where lower parts process the stimuli, and higher parts learn more general features. The neocortex consists of neurons, segments, and synapses. There are vertical connections that are the feedforward and feedback information between layers of cells and there are horizontal connections that are the context inputs. The neurons can connect to other nearby neurons through segments and synapses.

HTM is based on the core assumption that the neocortex stores and recalls sequences. These sequences are patterns of the Sparse Distributed Representation (SDR) input, which are translated into the sequences of cell activations in the network. This is an online training method, which doesn't need multiple epochs of training. Most of the necessary synapse connections are created during the first pass, so it can be viewed as a one-shot learning capability. The HTM network can recognize and predict sequences with such robustness, that it does not suffer from the usual problems hindering the training of conventional neural networks. HTM builds a predictive model of the world, so every time it receives input, it is attempting to predict what is going to happen next. The HTM network can not only predict the future values of sequences but e.g., detect anomalies in sequences.

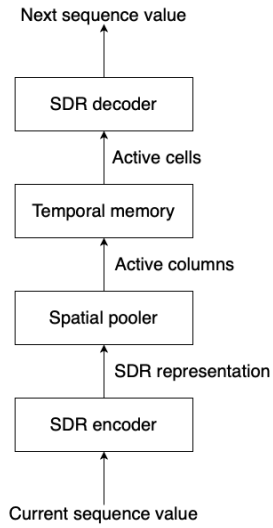

Fig. 1. HTM block diagram

The network consists of four components: SDR Encoder, Spatial Pooler, Temporal Memory, and SDR decoder (see Figure 1).

The four components do the following:

- The SDR Scalar Encoder receives the current input value and represent it an SDR. An SDR representation is a binary bit arrays that retains the semantic similarity between similar input values by overlapping bits.
- The Spatial Pooler activates the columns given the SDR representation of the input. The Spatial Pooler acts as a normalization layer for the SDR input, which makes sure the number of columns and the number of active columns stay fixed. It also acts as a convolutional layer by only connecting to specific parts of the input.
- The Temporal Memory receives input from the Spatial Pooler and does the sequence learning, which is expressed in a set of active cells. Both the active columns and active cells are sparse representations of data just as the SDRs. These active cells not only represent the input data but provide a distinct representation about the context that came before the input.

- The Scalar Decoder takes the state of the Temporal Memory and treating it as an SDR decodes it back to scalar values.

1) Sparse Distributed Representation: The capacity of a dense bit array is 2 to the power of the number of bits. It is a large capacity coupled with low noise resistance.

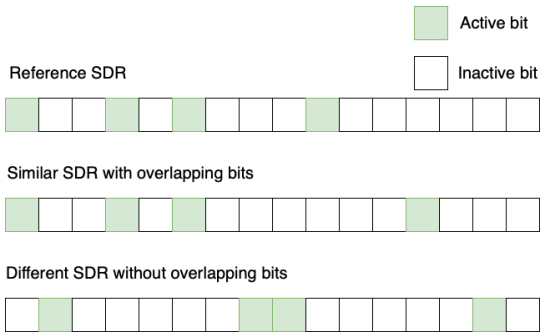

Fig. 2. SDR matching

A spare representation bit array has smaller capacity but is more robust against noise. In this case the network has a 2% sparsity, which means, that only the 2% of columns are activated [1]. A sparse bit array can be stored efficiently by only storing the indices of the ones.

To enable classification and regression there needs to be a way to decide whether or not two SDRs are matching. An illustration for SDR matching can be found in Figure 2. If the overlapping bits in two SDRs are over the threshold, then it is considered as a match. The accidental overlaps in SDRs are rare so the matching of two SDRs can be done with high precision. The rate of a false positive SDR matching is meager.

2) Encoder and decoder: The HTM network works exclusively with SDR inputs. There needs to be an encoder for it so that it can be applied to real-world problems. The first and most critical encoder for the HTM system is the scalar encoder. Such an encoder for the HTM is visualized by the Figure 3

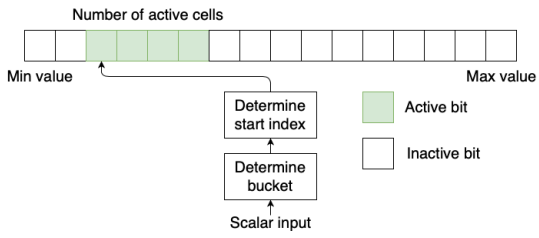

Fig. 3. SDR encoder visualization

The principles of SDR encoding:

- Semantically similar data should result in SDRs with overlapping bits. The higher the overlap, the more the similarity.
- The same input should always produce the same output, so it needs to be deterministic.
- The output should have the same dimensions for all inputs.
- The output should have similar sparsity for all inputs and should handle noise and subsampling.

The prediction is the task of the decoder, which takes an SDR input and outputs scalar values. This time the SDR input is the state of the network's cells in the Temporal Memory. This part of the network is not well documented, the only source is the implementation of the NuPIC package [18]. The SDR decoder visualization is presented in Figure 4.

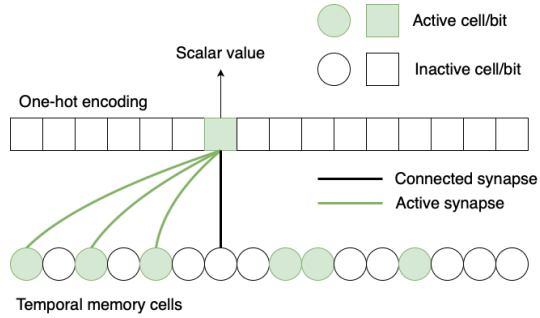

Fig. 4. SDR decoder visualization

3) Spatial Pooler: The Spatial Pooler is the first layer of the HTM network. It takes the SDR input from the encoder and outputs a set of active columns. These columns represent the recognition of the input and they compete for activation. There are two tasks for the Spatial Pooler, maintain a fixed sparsity and maintain overlap properties of the output of the encoder. These properties can be looked at like the normalization in other neural networks which helps the training process by constraining the behavior of the neurons.

The Spatial Pooler is shown in Figure 5

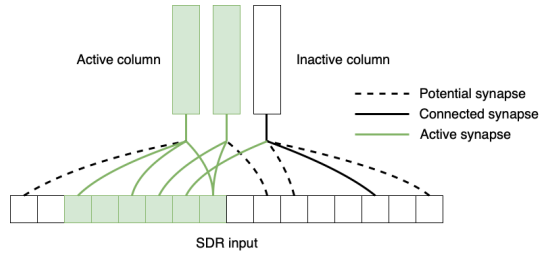

Fig. 5. Spatial Pooler visualization

The Spatial Pooler has connections between the SDR input cells and the Spatial Pooler columns. Every synapse is a potential synapse that can be connected or not depending on its strength. At initialization, there are only some cells connected to one column with a potential synapse. The randomly initialized Spatial Pooler already satisfies the two criteria, but a learning Spatial Pooler can do an even better representation of the input SDRs.

The activation is calculated from the number of active synapses for every column. Only the top 2% is allowed to be activated, the others are inhibited.

4) Temporal Memory: The Temporal Memory receives the active columns as input and outputs the active cells which represent the context of the input in those active columns. At any given timestep the active columns tell what the network sees and the active cells tell in what context the network sees it.

A visualization of the Temporal Memory columns cells and connections is provided in Figure 6.

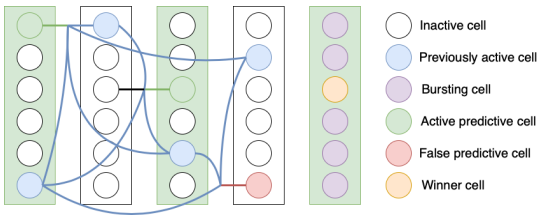

Fig. 6. Temporal Memory connections

The cells in the Temporal Memory are binary, active or inactive. Additionally, the network's cells can be in a predictive state based on their connections, which means activation is anticipated in the next timestep for that cell. The cells inside every column are also competing for activation. A cell is activated if it is in an active column and was in a predictive state in the previous timestep. The other cells can't get activated because of the inhibition.

The connections in the Temporal Memory between cells are created during training, not initialized like in the Spatial Pooler. When there is an unknown pattern none of the cells become predictive in a given column. In this case bursting happens. Bursting expresses the union of all possible context representation in a column, so expresses that the network does not know the context. To later recognize this pattern a winner cell is needed to choose to represent the new pattern the network encountered. The winner cells are chosen based on two factors, matching segments and least used cells.

- If there is a cell in the column that has a matching segment, it was almost activated. Therefore it should be the representation of this new context.
- If there is no cell in the column with a matching segment, the cell with the least segments should be the winner.

The training happens similarly to Spatial Pooler training. The difference is that one cell has many segments connected to it, and the synapses of these segments do not connect to the previous layer's output but other cells in the temporary memory. The training also creates new segments and synapses to ensure that the unknown patterns get recognized the next time the network encounters them.

The synapse reinforcement is made on the segment that led to the prediction of the cell. The synapses of that segment are updated. Also if there were not enough active synapses, the network grows new ones to previous active cells to ensure at least the desired amount of active synapses.

In the case where the cell is bursting the training is different. One cell must be chosen as winner cell. This cell will grow a new segment, which in turn will place the cell in a similar situation into the desired predictive state. The winner cell can be the most active cell, that almost got into predictive state, or the lowest utilized, in other words the cell that has the fewest segments. Only winner cells are involved in the training process. Correctly predicted cells are automatically winner cells as well, so those are always trained. The new segment will connect to some of the winner cells in the previous timestep.

5) Segments, synapses and training: In the HTM network segments and synapses connect the cells. Synapses start as potential synapses. This means that a synapse is made to a cell, but not yet strong enough to propagate the activation of the cell. During training, this strength can change and above the threshold the potential synapse becomes connected. A synapse is active if it is connected to an active cell.

The visualization for the segments connection to cells is provided in Figure 7 and the illustration for the synapses connecting to segments is in Figure 8.

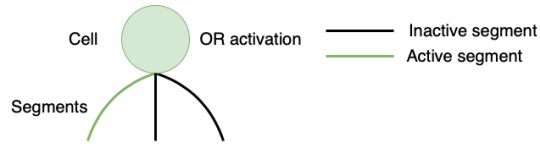

Fig. 7. Segment visualization

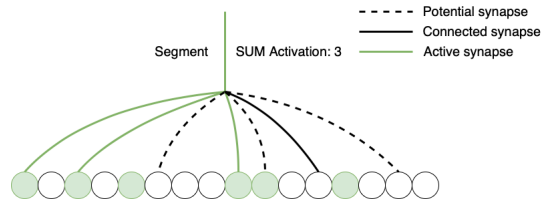

Fig. 8. Synapse visualization

Cells are connected to segments. The segments contain synapses that connect to other cells. A segment's activation is also binary, either active or not. A segment becomes active if enough of its synapses become active, this can be solved as a summation across the segments.

In the Spatial Pooler, one cell has one segment connected to it, so this is just like in a normal neural network. In the Temporal Memory, one cell has multiple segments connected to is. If any segment is activated, the cell becomes active as well. This is like an or operation between the segment activations. One segment can be viewed as a recognizer for a similar subset of SDR representations.

Training of the HTM network is different from other neural networks. In the network, all neurons, segments, and synapses have binary activations. Since this network is binary, the typical loss backpropagation method will not work in this case. The training suited for such a network is Hebbian learning. It is a rather simple unsupervised training method, where the

training occurs between neighboring layers only. The Hebbian learning is illustrated in Figure 9.

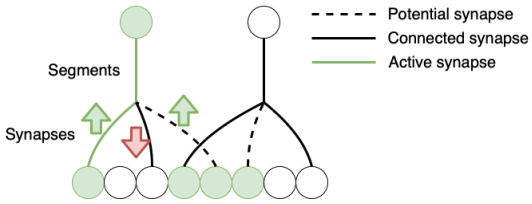

Fig. 9. Visualization of Hebbian learning

- Only those synapses that are connected to an active cell through a segment train.
- If one synapse is connected to an active cell, then it contributed right to the activation of that segment. Therefore its strength should be incremented.
- If one synapse is connected to an inactive cell, then it did not contribute right to the activation of that segment. Therefore its strength should be decreased.

C. HTM software solutions

There are HTM implementations maintained by Numenta, which give the foundation for other implementations.

First, the NuPIC Core (Numenta Platform for Intelligent Computing) is the C++ codebase of the official HTM projects. It contains all HTM algorithms which can be used by other language bindings. Any further bindings should be implemented in this repository. This codebase implements the Network API, which is the primary interface for creating whole HTM systems. It will implement all algorithms for NuPIC but is currently under transition. The implementation is currently a failing build according to their CircleCI validation [19].

NuPIC is the Python implementation of the HTM algorithm. It is also a Python binding to the NuPIC Core. This is the implementation we choose as baseline. In addition to the other repository's Network API, this also has a High-level API called the Online Prediction Framework (OPF). Through this framework predictions can be made and also it can be also used for anomaly detection. To optimize the network's hyperparameters swarming can be implemented, which generates multiple network versions simultaneously. The C++ codebase can be used instead of the Python implementation if explicitly specified by the user [18].

There is also an official and community-driven Java version of the Numenta NuPIC implementation. This repository provides a similar interface as the Network API from NuPIC and has comparable performance. The copyright was donated to the Numenta group by the author [20].

Comportex is also an official implementation of HTM using Clojure. It is not derived from NuPIC, it is a separate implementation, originally based on the CLA whitepaper [21], then also improved.

Comportex is more a library than a framework because of Clojure. The user controls simulations and can extract useful network information like the set of active cells. These variables can be used to generate predictions or anomaly scores.[4]

There are also unofficial implementations, which are based on the CLA whitepaper or the Numenta HTM implementations.

- Bare Bone Hierarchical Temporal Memory (bbHTM)[3]
- pyHTM[4]
- HTM.core[5]
- HackTM[6]
- HTM CLA[7]
- CortiCL[8]
- Adaptive Sequence Memorizer[9]
- Continuous HTM[10]
- Etaler[11]
- HTM.cuda[12]
- Sanity[13]
- Tiny-HTM[14]

III. PROPOSED METHOD

The goal of this work is to introduce sparse matrix operations to HTM networks to be able to realize larger models. Current implementations of the HTM network are not using sparse matrix operations, and these are using array-of-objects approach for storing cell connections. The proposed method is evaluated on two types of data: real consumption time-series and synthetic sinusoid data.

The first dataset is provided by Numenta called Hot Gym [22]. It consists of hourly power consumption values measured in kWh. The dataset is more than 4000 measurements long and also comes with timestamps. By plotting the data the daily and weekly cycles are clearly visible.

The second dataset is created by the timesynth Python package producing 5000 data points of a sinusoid signal with Gaussian noise.

A matrix implementation collects the segment and synapse connections in an interpretable data format compared to the array-of-objects approaches. The matrix implementation

[4]Comportex (Clojure), https://github.com/htm-community/comportex, Access date: 14th April 2020

[5]https://github.com/vsraptor/bbhtm, Access date: 14th April 2020

[6]pyHTM, https://github.com/carver, Access date: 14th April 2020

[7]htm.core, https://github.com/htm-community/htm.core, Access date: 14th April 2020

[8]HackTMM, https://github.com/glguida/hacktm, Access date: 14th April 2020

[9]HTM CLA, https://github.com/MichaelFerrier/HTMCLA, Access date: 14th April 2020

[10]ColriCl, https://github.com/Jontte/CortiCL, Access date: 14th April 2020

[11]Adaptive Sequence Memorizer, (ASM),https://github.com/ziabary/Adaptive-Sequence-Memorizer, Access date: 14th April 2020

[12]Continuous HTM GPU (CHTMGPU), https://github.com/222464/ContinuousHTMGPU, Access date: 14th April 2020

[13]Etaler, https://github.com/etaler/Etaler, Access date: 14th April 2020

[14]HTM.cuda, https://github.com/htm-community/htm.cuda, Access date: 14th April 2020

[15]Sanity, https://github.com/htm-community/sanity, Access date: 14th April 2020

[16]Tiny-HTM, https://github.com/marty1885/tiny-htm, Access date: 14th April 2020

achieves the same functionality as the baseline Numenta codebase. The dense matrix implementation has a massive memory consumption, that limits the size of the model. Sparse matrix realization should decrease the required amount of memory.

However, porting to a sparse solution is not straightforward since the support of efficient sparse operations is far less than regular linear algebra.

A. System design and implementation

In this section the implemented sparse HTM network design is introduced. Throughout the implementation the sparse Python package Scipy.sparse was used, so first that package is described in detail. Next, the four layers of the network are presented, namely the SDR Scalar Encoder, the Spatial Pooler, the Temporal Memory, and the SDR Scalar Decoder. The detailed sparse implementations of these submodules are described, with the matrix implementation in the focus. In this part the sparse matrix and sparse tensor realizations are also discussed.

B. Matrix implementation

As an initial step, a dense matrix implementation of the NuPIC HTM network was designed and created, which allows treating the HTM network the same as the other widely used and well-optimized networks. While this step was necessary for comparison, it is not suitable for large dataset, since the size of these networks is much bigger compared to the LSTM or CNN networks.

1) Spatial Pooler: The network interprets every input into SDR representations which are represented as binary vectors. Multiple inputs can be represented as binary matrices.

The Spatial Pooler columns are connected to the SDR encoder through one segment and its synapses. These connections can be expressed with a matrix, where every row represents an input cell and every column represents a Spatial Pooler column.

The synapse connections have strengths but are used in a binary fashion based on synapse thresholds. Using the binary input vector and the binary connection matrix the column activations are calculated using matrix multiplication. The active column are the ones with the top 2% activation.

2) Temporal Memory: In the Temporal Memory cells are connected with other cells. In addition one cell can have multiple segments, so there needs to be a matrix representing every cells connections. For all the cells this results in a tensor, the dimensions are the number of cells along two axis, and the number of maximal segments per cell.

The calculation of cell activation has an extra step, because of the multiple segments. First, the segment activation are calculated for every cell using binary matrix multiplication just as in the Spatial Pooler. These results combined are a matrix, which dimensions are the number of cells times the number of segments. After the activations are calculated, those segments are activated that have above threshold activation values. This results in a binary matrix. Then the cells that are set to be in predictive state are the ones with at least one active segment, with an or operation along the segment axis.

C. Sparse implementation

Using sparse matrices enables to better scale the network compared to the dense matrix representation. In order to introduce sparse matrix operations to the HTM we used the Scipy.sparse Python package. However, there are missing tensor operators, which were required to be implemented.

There are multiple ways of implementing a sparse matrix representation – different formats can be used for different use-cases. There is the compressed sparse row representation (CSR). The row format is optimal for row-based access in multiplying from the left. The pair of this format is the compressed sparse column format (CSC), which is optimized for column reading, like in right multiplication. From these, the linked list format is beneficial, because it enables the insertion of elements. In the other two cases, insertion is a costly operation.

1) Sparse matrix: The network uses the Scipy.sparse package as the main method for matrix operations. This package is extended for further use in the HTM network and also to implement the sparse tensor class. It involves all the common sparse matrix format, which are efficient in memory complexity and have small overhead in computational complexity. This computational complexity decreases as the matrix becomes sparse enough (for matrix dot product around 10% is the threshold).

The realized SparseMatrix class is a wrapper for the Scipy.sparse Python module, extended with operators needed for the Spatial Pooler like reshaping, handling of binary activation matrices, logical operators along axis and random element insertions.

2) Sparse tensor: Scipy does not have a sparse tensor implementation. In our case the solution is a dictionary of sparse matrices stacked together. The third dimension is also sparse, it only has a sparse matrix at a given index if it contains at least one nonzero value. The SparseTensor class uses the SparseMatrix class, implementing the same operators.

After all the sparse implementation of the Spatial Pooler and the Temporal Memory differ only in the used classes, since the interfaces are shared across the two. The Spatial Pooler uses sparse vectors as inputs and sparse matrices to store and train the connections. The Temporal Memory receives the input in sparse vectors and stores and trains the connections using sparse tensors.

IV. EVALUATION AND RESULTS

We carried out experiments on two levels: on operation and network levels. On operation level the dense and sparse realizations were compared in terms of speed and memory capacity, while on the network level the training times and modeling performance of different architectures were compared. In the latter case four different networks were investigated: LSTM, NuPIC HTM, dense HTM, and sparse HTM networks. The baseline LSTM network consists of an LSTM layer with 100

cells and a dense layer also with 100 cells. The training data was generated in autoregressive nature, i.e. with a receptive field of 100 timesteps the network should predict the next instance. The NuPIC HTM network consists of four modules: an SDR Encoder, a Spatial Pooler, a Temporal Memory, and an SDR Decoder(/Classifier). These are configured as the default sizes by Numenta as having 2048 columns, 128 cells per column and 128 maximum segments per cell in the Temporal Memory.

A. Performance test of the operations

In order to understand the efficiency of the sparse implementation we investigated the scenarios in which the HTM network utilizes sparse matrices. That involves the creation of matrices, element wise product, dot-product, addition, subtraction, and greater or less than operations. The measurements were carried out using randomly generated matrices and tensors at fixed sparsities.

The measurements shown in Table I are carried out on CPU (Intel Core i5, 2 cores, 2GHz)and each represent an average of 1000 runs. Both the dense and sparse matrices are 1000x1000 in size with sparsity of 0.1%.

TABLE I
DENSE AND SPARSE ARITHMETIC MATRIX OPERATION EXECUTION TIMES ON CPU (1000 SAMPLE AVERAGE)

Operation	Dense time	Sparse time
Addition	0.001080s	0.000418s
Subtraction	0.001104s	0.000413s
Dot-product	0.0545s	0.0012s
element wise product	0.001074s	0.000463s
Greater than	0.000672s	0.000252s
Less than	0.000649s	0.049147s

It is clear that the sparse version has an advantage for almost all operators, however, the ”less than” operator lags behind compared to the dense counterpart. This is because the result has all the values set to true, which is not ideal for a sparse representation. (true being a nonzero value) Still the execution stores this as a sparse matrix which has a measurable overhead.

Next, to understand the efficiency of the sparse tensors we measured the actual scenarios in which the HTM network uses sparse matrices. That is the creation of tensors, element wise product, dot-product, addition, subtraction, and greater or less than operations.

The measurements are shown in Table II with the same settings as before. Both the dense and sparse tensors' shape is 10x1000x1000 with sparsity of 1%.

These results show, that the sparse tensor is slower in cases of addition and subtraction and element wise product. However, this solution excels in dot product, which is the operator needed to calculate activations for a given input. The speedup here compared to the dense implementation is more than a 1000 times. In the case of the less than operator the implementation is also slower due to the same reasons as with the sparse matrices.

TABLE II
DENSE AND SPARSE ARITHMETIC TENSOR OPERATION EXECUTION TIMES ON CPU (1000 SAMPLE AVERAGE)

Operation	Dense time	Sparse time
Addition	0.0040s	0.0079s
Subtraction	0.0037s	0.0069s
Dot-product	33.29s	0.0262s
element wise product	0.0034s	0.0074s
Greater than	0.0022s	0.0038s
Less than	0.0015s	0.3393s

B. Performance test of HTM modules

In Table III each different module of the two networks were measured on CPU. Each measurement represent an average over 100 samples.

TABLE III
NUPIC AND SPARSE HTM MODULE EXECUTION TIMES ON CPU (100 SAMPLE AVERAGE)

Part of the network	NuPIC	Sparse HTM
SDR Encoder	0.000303s	0.000607s
Spatial Pooler	0.0179s	0.0139s
Temporal Memory	0,0136s	1.03s
SDR Decoder	0.000303s	0.24s

These results show that the proposed sparse implementation has an advantage in the Spatial Pooler, where the inference is more straightforward, so the sparse multiplication speedup can be utilized. However, the Temporal Memory solution still lags behind in execution time.

For the memory complexity the following network sizes are applicable. These numbers are recommendations of Numenta, and are based on their research. The number of columns is a minimum requirement because of the sparse activation. The other parameters have a specific capacity that can be further fitted to a specific need. In general these values should work without the network becoming too big in size and capacity. The SDR size should be 100 with 9 active elements, the network size is 2048 columns, 32 cells per column and 128 segments per cell. The activation should be 40 columns in each timestep. The values in Table IV are measured in the number of integer values stored.

TABLE IV
MEMORY COMPLEXITY OF THE DIFFERENT NETWORK PARTS

Part of the network	Dense HTM	Sparse HTM
SDR Encoder output	100	27
Spatial Pooler connections	204800	12000
Spatial Pooler output	2048	120
Temporal Memory connections	$5.49 * 10^{11}$	$3.35 * 10^{8}$ (max)
Temporal Memory output	65536	40-1280
SDR Decoder connections	6553600	65536 (max)

It is clear that, the sparse solution makes it possible to store the network in a matrix format, since the Temporal Memory

part can easily exceed current hardware limitations. In the case of NuPIC HTM the memory complexity estimation is harder, since that uses array-of-objects structure.

TABLE V
TRAINING TIMES

Configuration	Epochs	Timesynth	Hot Gym
LSTM	100 epochs	243s	318s
NuPIC HTM	1 epoch	130s	138s
NUPIC HTM	5 epochs	1706s	1028s

Last, the LSTM and HTM network predictions for different datasets were investigated. The training times are shown in Table V for the LSTM and NuPIC HTM solutions.

First, in Figure 10 the predictions for both LSTM and HTM can be seen on the test part of Hot Gym dataset for the first 100 elements. The y-axis represents the power consumption of the building. The figure presents the difference between the LSTM and HTM predictions, where LSTM is less noisy and it seems that it gives near naive predictions in some cases. The HTM tends to follow better rapid changes.

In the case of train and test losses the HTM network is not on par with the performance of the LSTM network. In Table VI the performances are evaluated using the Hot Gym test dataset. It shows that the LSTM maintains lower train and test loss values than the HTM network. However, based on the possible interval range and looking at the predictions it is not clear that the HTM network is worse at predicting this dataset (see Figure 10). On the other dataset the achieved results are summarised in Table VII. In this case also the LSTM has a lower training and testing mean squared error. Looking at the predictions the network completely filters out the high frequency part of the data and only retains the base sinusoid signal in the predictions (see Figure 11).

TABLE VI
MINIMUM MSE VALUES FOR HOT GYM DATASET

Configuration	Epochs	Train MSE	Test MSE
LSTM	100	0.1073	0.1658
NuPIC HTM	5	0.3762	0.4797

TABLE VII
MINIMUM MSE VALUES FOR TIMESYNTH DATASET

Configuration	Epochs	Train MSE	Test MSE
LSTM	100	0.1603	0.1474
NuPIC HTM	5	0.4940	0.5069

In Figure 11 the predictions for both networks can be seen on the synthetic (Timesynth) test dataset for the first 200 timesteps. In this case it is even more pronounced that the LSTM network smooths the rapid changes in its predictions. There is also a lag its predictions, that is seen as a shift in the direction of the x-axis.

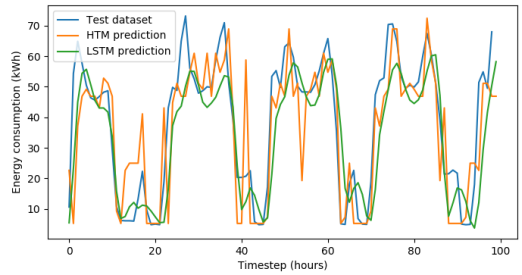

Fig. 10. Predictions for Hot Gym dataset

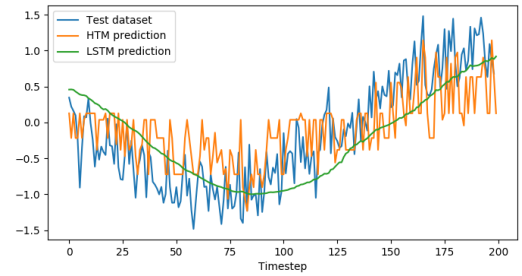

Fig. 11. Predictions for synthetic (Timesynth) data

In Figure 12 the outputs of HTM network are presented after the first and last epoch of training on the Hot Gym test dataset. It shows that the network is able to follow the main cycles of the data from the first epoch. On the other hand in Figure 13 this kind of progress is not that clear.

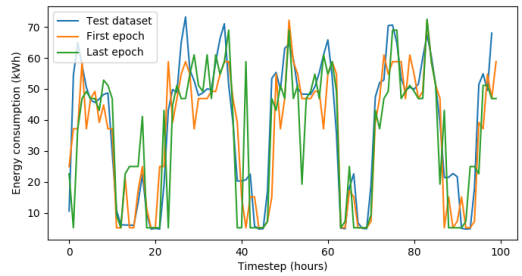

Fig. 12. Difference between first and last epoch for HTM on Hot Gym dataset

V. CONCLUSIONS

In this paper we investigated the sequence learning possibilities of HTM network. The advantages and disadvantages of different implementations surrounding the HTM network were described and different HTM versions and an LSTM were evaluated on a synthetic sequential dataset and on a real time-series. A methodology of turning the implementation of the HTM network to sparse matrix operations was proposed for lower memory usage. We showed that the proposed methodology is feasible, it uses at least an order of magnitude less

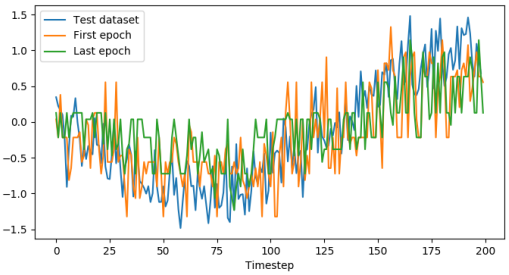

Fig. 13. Difference between first and last epoch for HTM on synthetic (Timesynth) dataset

memory than the dense implementation in the case where the sparsity of the network is at 2%. Furthermore, the proposed method's performance remains comparable to the other HTM implementation.

ACKNOWLEDGMENT

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications), by the BME-Artificial Intelligence FIKP grant of Ministry of Human Resources (BME FIKP-MI/SC), by Doctoral Research Scholarship of Ministry of Human Resources (ÚNKP-19-4-BME-189) in the scope of New National Excellence Program and by János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

REFERENCES

[1] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, "Biological and machine intelligence (bami)," 2016, initial online release 0.4. [Online].Available: http://numenta.com/biological-and-machine-intelligence/

[2] Numenta, "Numenta webpage," https://numenta.com, 2019.

[3] ——, "Htm school," https://www.youtube.com/playlist?list=PL3yXMgtrZmDqhsFQzwUC9V8MeeVOQ7eZ9, 2018.

[4] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[5] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling," in *Thirteenth annual conference of the international speech communication association*, 2012.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, nov 1997. [Online]. Available: DOI: 10.1162%2Fneco.1997.9.8.1735

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[8] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361–365, 1996.

[9] J. Contreras, R. Espinola, F. Nogales, and A. Conejo, "ARIMA models to predict next-day electricity prices," *IEEE Transactions on Power Systems*, vol. 18, no. 3, pp. 1014–1020, aug 2003. [Online]. Available: DOI: 10.1109%2Ftpwrs.2002.804943

[10] P. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990. [Online]. Available: DOI: 10.1109%2F5.58337

[11] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," in *Neural networks: Tricks of the trade*. *Springer*, 2012, pp. 9–48.

[12] Y. LeCun, Y. Bengio et al., "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[13] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, apr 1998. [Online]. Available: DOI: 10.1142%2Fs0218488598000094

[14] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber, "A clockwork rnn," *arXiv preprint arXiv:1402.3511*, 2014.

[15] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," *arXiv preprint arXiv:1609.01704*, 2016.

[16] S. Merity, "Single headed attention rnn: Stop thinking with your head," arXiv preprint arXiv:1911.11423, 2019.

[17] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. *Association for Computational Linguistics*, 2016. [Online]. Available: DOI: 10.18653%2Fv1%2Fn16-1174

[18] Numenta, "Nupic (python)," http://github.com/numenta/nupic, 2019.

[19] ——, "Nupic core (c++)," http://github.com/numenta/nupic.core, 2019.

[20] ——, "Htm.java," http://github.com/numenta/htm.java, 2019.

[21] ——, "Htm cortical learning algorithms," https://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf, 2011.

[22] ——, Hot Gym dataset, 2024 (accessed July 1, 2020). [Online]. Available: https://github.com/numenta/nupic/blob/master/examples/opf/clients//hotgym/prediction/one_gym/rec-center-hourly.csv

IEEEbiograph

Csongor Pilinszki-Nagy conducts research on artificial general intelligence methods since 2016. His work consists of solutions using image recognition by convolutional neural networks, building general solutions in game environments using reinforcement learning and researching unconventional methods like Hierarchical Temporal Memory network. He obtained his MSc degree in Computer Science from the Budapest University of Technology and Economics in January 2020. He is a member of the Balatonfüred Student Research Group.

Bálint Gyires-Tóth conducts research on fundamental and applied machine learning since 2007. With his leadership, the first Hungarian hidden Markovmodel based Text-To-Speech (TTS) system was introduced in 2008. He obtained his PhD degree from the Budapest University of Technology and Economics with summa cum laude in January 2014.
Since then, his primary research field is deep learning. His main research interests are sequential data modeling with deep learning and deep reinforcement learning. He also participates in applied deep learning projects, like time series classification and forecast, image and audio classification and natural language processing. He was involved in various successful research and industrial projects, including finance, fraud detection and Industry 4.0. In 2017 he was certified as NVidia Deep Learning Institute (DLI) Instructor and University Ambassador.