# Approve prediction for multisequence learning

Poonam Dashrath Paraskar
poonam.paraskar@stud.fra-uas.de

Pratik Prakash Desai
pratik.desai@stud.fra-uas.de

Ankita Talande
ankita.talande@stud.fra-uas.de

***Abstract*— This paper delves into Multisequence Learning, a technique designed for acquiring insights into sequence learning and prediction. An examination of the current Multisequence Learning implementation is conducted to comprehend the mechanisms involved in learning and predicting sequences. This approach used to investigate implicit learning, where the model designed to learn the pattern in a few iteration steps and to generate the Sparse Distributed Representation (SDR) of the input and provide the matching sequences as the predicted output using predictor.**

***The existing implementation deals with training of input sequences but the process of reading input data was through hardcoded values in the codebase, whereas new implementation utilizes current working directory path for storing the input files and automates the input reading phase. This experiment has variety of industrial applications such as speech recognition, robotics natural language processing, time series prediction, financial engineering, and DNA sequencing.***

***The paper highlights the new improvements made to existing application and newly introduced accuracy calculation phase for the predicted sequence. Accuracy of the predicated sequence is calculated based on matching input digits with the predicted digit of the SDR and number predictions made. Accuracy can be defined as presence of specific value of test data in the given input dataset, and it is denoted in percentage. In general, the study offers insights into multi-sequence learning, introduces a novel technique that can be utilized to enhance the precision and effectiveness of sequence learning and prediction, and stores the final accuracy result in the external file.***

***Keywords—: Machine Learning, multisequence learning, SDR, Prediction, Accuracy.***

## I. Introduction

Hierarchical temporal memory (HTM) is a type of machine learning model inspired by the structure and function of the neocortex in the human brain. Its purpose is to learn sequences of information and make predictions based on them. The HTM model consists of several components, including an encoder, a spatial pooler, a temporal memory, and an HTM classifier. Each element in a sequence is represented by a Sparse Distributed Representation (SDR). The temporal memory learns the sequences of these SDRs and makes predictions about future input SDRs. [1]

During the learning phase, the temporal memory predicts the next element in the sequence at every step.
After the last prediction, a learning cycle ends, and the temporal memory starts again with the same sequence.
The goal is for every cycle to have correct predictions for every element, resulting in an accuracy of 100%. The aim is to achieve 100% accuracy for as many consecutive cycles as possible.
Sequence-to-sequence learning is a general framework used for machine translation, where the encoder maps the input sequence to a fixed-length vector. [2]

## II. LITERATURE SURVEY

Components utilized for experiment.

1. SDRs
2. HTM (Hierarchical Temporal Memory)
3. Spatial Pooler
4. Encoders
5. Predictor
6. Result of Training Phase.

*1.* ***SDRs****:* Sparse Distributed Representation (SDR), a system for information organization in HTM, is effective. A small portion of the big, interconnected cells are only partially active at any given time, which is referred to as "sparse" in this context. The word "distributed" means that the active cells, which are used to represent the activity of the region, are dispersed throughout the territory. HTM employs a binary SDR, which comes from a particular encoder and is more computationally and biologically reasonable. Because to the binary SDR's crucial properties, functional information is not lost even though the number of possible inputs surpasses the number of possible representations. [3]

2. ***HTM:*** Hierarchical temporal memory (HTM) provides a theoretical framework that models several key computational principles of the neocortex. The HTM architecture is made up of layers of hierarchically stacked processing nodes, also referred to as neurons. Each neuron interacts with other neurons in a dispersed, sparse network to process the input it receives. The HTM architecture is designed to perform pattern identification, anomaly detection, and prediction tasks using time-series data. It accomplishes this by using a memory approach called temporal pooling, which gives it the ability to learn and recognize patterns over time even in the presence of noise and variance. [4]

   Information is transferred up the hierarchy in layers in the HTM design, with each layer producing progressively more intricate representations of the input. An HTM system typically receives its input as a stream of high-dimensional sensory data, such audio or video. Depending on the input, the output consists of several predictions or actions. Overall, the HTM architecture provides a solid and flexible framework for building intelligent systems that can adapt to and learn from new data over time. [4] This makes it suited for a wide range of applications, including robotics, computer vision, and natural language processing.

3. **Spatial Pooler***:* By assigning active cells to columns, the Spatial Pooler creates a Sparse Distributed Representation (SDR) input. Synapses connect each column to the next region of input bits; despite the fact that multiple columns may look same, they are all distinct from one another. Various patterns generate different levels of activation, and in the columns, stronger activation suppresses weaker activation levels. The size of the columns can be changed to accommodate little or huge areas. In order to restrict the representation of input, an inhibitory mechanism is put in place. The HTM develops connections between cells based on the input. Synapse permanence updating is a type of learning. Whereas inactive columns have a lower persistence value, active columns have a higher one. [5]

4. **Encoders***:* An HTM's evolution is influenced by the data it receives and the way that data is shown. An encoder is used to transform arbitrary input into a format that an HTM can comprehend, allowing the HTM to interpret the input. Each bit in this Sparse Distributed Representation (SDR) format indicates the activation state of a column in the HTM's prior area. The following region of the HTM then uses the SDR as a feedforward input. [6]

5. **Predictor:** Predictor class aids in the computation of predictions used for sequence learning. The algorithm returns an instance of this class following the learning process. A method with a list of input components is provided by this class. The predictor tries to anticipate the following element for each input element that is displayed. The predictor responds with a greater score the more elements that are presented in a sequence.

   The Predictor object is returned by the Run() method of the MultiSequenceLearning class. This Predictor object is an instance of the PredictorBase abstract class defined in the NeoCortexApi library.
   The Predictor object returned by the Run() method is an instance of a specific subclass of PredictorBase, which is responsible for making predictions based on the input sequences that were used to train the model. The exact subclass of PredictorBase that is returned depends on the specific type of prediction task that is being performed.
   The PredictorBase class provides a common interface for making predictions across different types of prediction tasks, such as classification, anomaly detection, or sequence prediction. It defines a Compute() method that takes an input vector and returns a list of predicted values. The specific implementation of Compute() depends on the subclass of PredictorBase.
   The Predictor object is used to make predictions about future values in the input sequences after the Model has been trained on historical data. The Predictor object encapsulates the internal state of the trained model and is used to generate predictions by computing the most likely next value in the sequence based on the current input and the model's learned patterns.

6. **Training phase:**

Model is getting trained for each input sequence and newborn cycle gets generated till the spatial pooler reach the stable state.

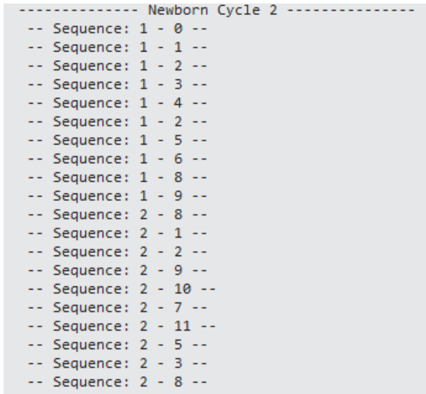



*Figure 1.1 Training Phase*

## III. METHODS

Before deep dive in the method first understand datasets which are used for Multi Sequence Learning.

***Datasets***

Multiple sequence of numbers in csv-file as below:

seq1: 2, 4,6,8,10,12,14,16
seq2: 3, 7, 14, 16,22,25,30
seq3: 1, 2,3,4,5,6,7,8,9,12
.
.
Seqn: 2,5,7,3,1,6,7,8,9,13

1. *GetInputFromTextFile( ):*

   *Issue:* In the existing implementation of Multisequence learning project, in all the methods inputs were hardcoded so if user wants to change the input, then it’s necessary to change the input sequences from the code. so, to resolve the issue we have tested different methods to take the inputs from the file.
   *Solution:* Team has implemented logic to take the inputs from the Text file. We have tried 2 approaches to split the multiple input sequences by using comma ‘,’ to separate each digit of the input sequence and using special character at the end of each sequence for splitting it from other input sequences. In this case we used semi-colon ‘;’ to split.

2. *GetInputFromCsvFile( ):*

   *Issue:* The significant issue we faced by using the .txt file approach is we had to add both comma ‘,’ and semi-colon ‘;’ at the end of each input sequence, which is not a feasible solution and by which text file also looks inappropriate.
   *Solution:* We have implemented .csv file logic to take the inputs from the CSV file. CSV stands for "Comma-Separated Values". It is a file format used for storing and exchanging tabular data, such as spreadsheets or databases. In a CSV file, each line represents a row of data and each field within a row is separated by a comma. CSV files are simple and widely supported, making them a popular choice for data exchange between different systems and applications.

3. *GetInputFromExcelFile():*

   *Issue:* The problem with CSV file is we need to add one non double character at end of each row to terminate the row/sequence and take the next sequence. This can cause an issue in real time working environment.
   *Solution:* In implemented method we are using .xlsx file type to take the input sequences. Which are referred as training data sequences. Here we overcame the issues of the previous methods where we need to add any non-double value to terminate the row/sequence and to jump to the next row/sequence and any special in case of text file to jump over the next input sequence. To implement this feature we used the string.IsNullOrWhiteSpace( ) property.

4. *GetSubSequencesInputFromExcelFile( ):*

   *Issue:* In the existing implementation inputs were hardcoded so if user wants to change the input, then it’s necessary to change the input sequences from the code.
   *Solution:* Team has implemented the subsequence logic to take the subsequence test input from the .xlsx file. We are passing the TestSubSequences to the SubSequences list of type double. After reading all the TestSubSequences we are returning SubSequences.

5. *Accuracy Logs:* Team used StreamWriter() class to create the file. If the file exists, it can be overwritten or appended to. If the file does not exist, this constructor creates a new file. The true flag appends to the file instead of overwriting it. Here we are generating logs for sequenceKeyPair.Key and accuracy. Ex. Sequence: 1 is having accuracy 30%

6. *Encoder Settings:* For encoder settings we have modified the value from 0-99. We have added input validation for the same in our program.cs file.

7. *FilePath:* For reading inputs, test subsequences and writing accuracy to .xlsx file we are using the relative filePath of the input file, subsequence file and Final Accuracy file. We are using Environment.CurrentDirectory property in C# that gets or sets the current working directory of the application. It returns a string that represents the absolute path of the current working directory.

8. *Final Accuracy:* We have implemented a logic that writes the final accuracy in .xlsx file along with the current date and time so if we need to check the accuracy in future we can get to know when the file is generated. For that we are using DateTime.Now property in c# that returns the current local date and time on the computer where the code is running. It returns a DateTime object that contains the current date and time.

   *string fileName = string.Format("Final Accuracy ({0:dd-MM-yyyy HH-mm-ss}).csv", DateTime.Now);*

## IV. Implementation

This part of the report explains stages of the experiment. This experiment broadly carried out into two stages.

1. Learning/Training phase
2. Prediction and Accuracy calculation phase.

1. **Learning phase/Training Phase:** In learning phase input data sequences are getting passed to *RunExperiment()* method. In *RunExperiment()* method training of input sequences is done using Cortex Layer, Spatial Pooler, Homeostatic Plasticity Controller which checks the stability of spatial pooler. Training of input sequences is required to get the stable state of Spatial pooler. Newborn cycles are generated for each input sequence till the time Spatial pooler reach the stable state. In newborn cycle, compute method of Cortex Layer is getting executed. Once Spatial pooler reaches to stable state the Temporal Memory algorithm is getting activated. At this stage, Spatial Pooler is trained completely. With pretrained SP and HPC, the TM learn cells for patterns.
   The figure 1.2 explains how training phase is carried out:

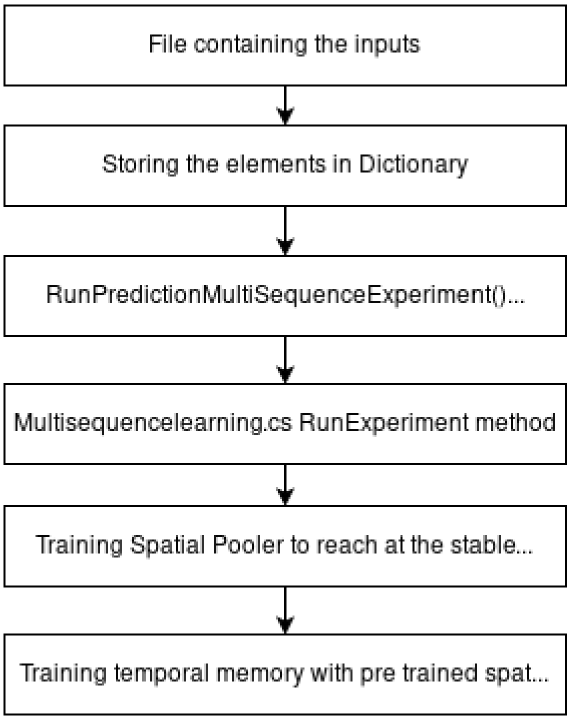



*Figure 1.2 Training Phase*

2. **Prediction and Accuracy calculation phase:** *PredictNextElement()* method and Predictor class is used for prediction.
   After the learning process, the algorithm returns the instance of Predictor class. This class provides *Predict()* method with a list of input elements.
   For every presented input element, the predictor tries to predict the next element.
   The more element provided in a sequence the predictor returns with the higher score then model produces a similarity matrix for all the classes.
   After prediction of test sequence and next element, accuracy calculation is done.

   The figure 1.3 explains how prediction phase is carried out:

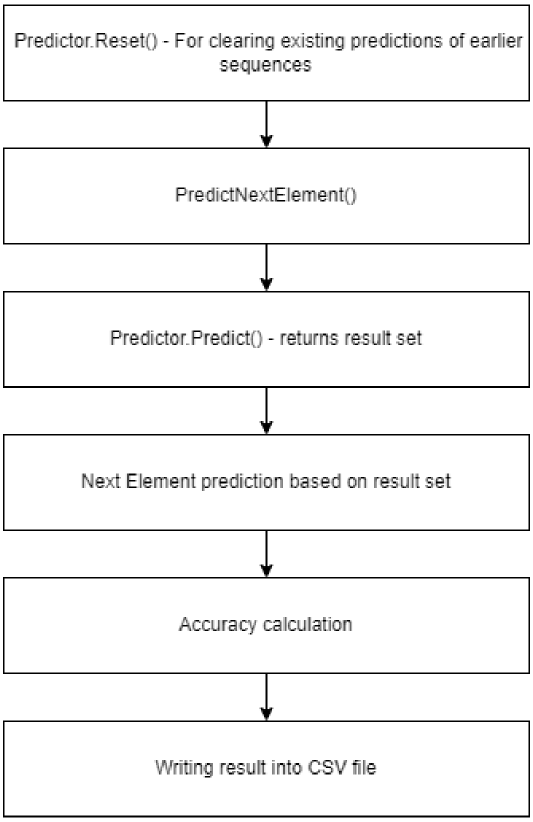


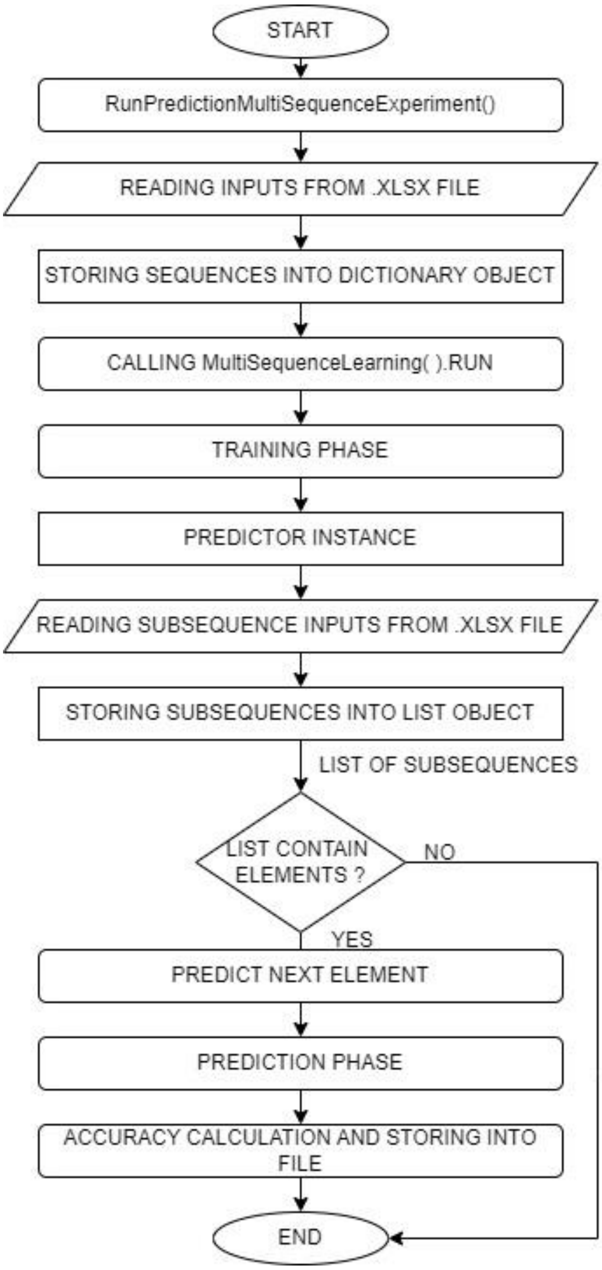

*Figure 1.3 Prediction Phase*

## V. RESULTS

We have divided our result into six sections.

1. Algorithm/Flow chart of the code
2. Result of training phase
3. Result of prediction phase
4. Result of accuracy calculation phase
5. Exporting existing accuracy result into .csv file
6. Performance testing and comparison with legacy code

1. Flow chart of the code:
   Figure 1.4 explains the flowchart of MultiSequenceLearning experiment.
   This figure shows a step-by-step explanation of the workflow. Flow chart begins with the input reading phase and it has been modified for the new implementation.
   It is followed by the training phase, which is the same as the prior version.
   Following the completion of the training phase, the flow moves to the prediction phase, during which the predictor predicts the next element in the sequence.
   The last step is accuracy calculation phase, which has just been added as a part of new implementation and the final accuracy calculation results are being saved in an external file.




```
Cell SDR: 2562, 2721, 2795, 28
Match. Actual value: Sequence:
NO CELLS PREDICTED for next cy
Sequence: 3 is having Accuracy
Cycle: 331  Matches=4 of 5   8
100% accuracy reched 28 times.
```

*Figure 1.4 Algorithm of Code.*

2. Result of prediction phase.

```
Col  SDR: 245, 263, 280, 29
Cell SDR: 6144, 6587, 7024,
Missmatch! Actual value: Se
Item length: 20  Items: 21
```

Prediction phase is carried out once training phase is completed. Prediction phase executes in three steps match detection, mismatch detection and next element prediction.

*Figure 1.5 Match detection*

If the input sequence and test sequence are matching then predict element will detect a match and match counter is getting incremented. Which we are using for accuracy calculation. Figure 1.5 is showing the match.

*Figure 1.6 Mismatch detection*

If any element of the input sequence does not match with the test data predictor will return a mismatch. The figure 1.6 is showing the mismatch.

3. Result of Accuracy Calculation Phase

Accuracy is calculated by dividing matches with total number of sequenceKeyPair multiply by 100.

*Accuracy = count of matches / total number of predictions * 100*

By performing this calculation, we are getting accuracy in percentage. This accuracy we are storing into the csv file and also printing this information into the debug window as shown in the

*Figure 1.7 Accuracy Calculation*

4. Exporting existing accuracy result into .csv file

   In MultiSequenceLearning.cs class accuracy is getting calculated by using formula
   *double accuracy = (double)matches / (double)sequenceKeyPair.Value.Count * 100.0;*
   This accuracy result we are exporting into csv file which is available in MySeProject. [8]

5. Performance testing and comparison with legacy code.
   In this phase of testing, we tested the code for 10 iterations with different range of training data and test data and compare the performance of newly implemented the code and legacy code. Result can be found in attached excel sheet. [9]

## VI. CONLUSION

This project described that the Neocortex API's Multi Sequence learning reference model was used to perform multi-sequence learning for a sequence of numerical data sets. The HTM Prediction Engine was adjusted with various parameters to optimize the training process, and the data was transformed into an encoded value and stored in a dictionary using an Encoder and SDR input for training.

Team has developed new method that read learning sequences from a file by itself and learn them. After that the sample should read testing subsequences from another file and calculate the prediction accuracy.

Different methods are created to predict the trained sequences by comparing the generated similarity matrix with each of the SDRs of the learned sequence from the training phase, and the accuracy percentage of the predicted sequences was calculated and stored in a file. The accuracy of the predictions was found to increase with the number of cycles, indicating that the learning process improved over time.

This project provided insights into various aspects of the Neocortex API, including the functioning of encoders, how the Spatial Pooler generates SDR inputs and performs the learning phase in stabilizing the learning process.

## VII. REFEERENCES


[1] Cortical.io. [Online]. Available: https://www.cortical.io/science/sparse-distributed-representations/.

[2] S. A. a. J. H. Y. Cui, "The HTM Spatial Pooler —A Neocortical Algorithm for Online Sparse Distributed Coding," Vols. 11-2017, November 2017,.

[3] S. A. a. J. H. Yuwei Cui, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," 29 November 2017. [Online]. Available:https://www.frontiersin.org/articles/10.3389/fncom.2017.00111/full.

[4] A. G. B. W. T. D. D. Pech, "Scaling the HTM Spatial Pooler,"*International Journal of Artificial Intelligence & Applications*, vol. 11, 2020.

[5] [Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Accuracy_Testing.xlsx.

[6] [Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20Testing.xlsx.

[7] P. Paraskar and P.Desai.[Online].Available: https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20Testing.xlsx.

[8] "https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Accuracy_Testing.xlsx," [Online].

[9] "https://github.com/pparaska/neocortexapi_Team_MSL/blob/Team_MSL/source/MySEProject/Performance%20Testing.xlsx," [Online].