Information Technology Course
Module Software Engineering
by Damir Dobric / Andreas Pech

Enhancing Multi Sequence Learning with Accuracy

Syeda Ayesha Asad
syeda.asad@stud.fra-uas.de

Muhammad Zubair
muhammad.zubair@stud.fra-uas.de

Syed Tabish Talha Hassan
syed.hassan@stud.fra-uas.de

Abstract- In practical scenarios, the anticipation and comprehension of temporal sequences from sensory inputs play a crucial role. Drawing upon various characteristics of neurons, the Hierarchical Temporal Memory (HTM) framework offers a theoretical basis for sequence learning. Emulating the operational principles of the neocortex, HTM facilitates the learning and storage of sequential patterns in memory, enabling predictive operations until an appropriate match is attained. This study assesses the performance of the HTM Prediction Engine across alphabetic and number sequences. The principal aim is to investigate the capabilities of the HTM Prediction Engine and comprehend the concept of multi-sequence learning concerning alphabetic and number sequences.

Keywords— Hierarchical Temporal Memory (HTM), Sequence Learning, HTM Prediction Engine, Spatial Pooling.

I. INTRODUCTION

The process of learning and predicting sequences holds potential for various applications, such as music note recognition, gene detection, identification of regulatory regions, and early detection of diseases. This project aims to enhance a previously developed multi-sequence learning model. This includes implementing features like automatic data generation, simultaneous training with multiple datasets, adapting alphabet datasets to the existing model, and enhancing accuracy through parameter adjustments.

Data generation is achieved using a sequence generator capable of creating numeric and alphabetic sequences based on user-provided parameters. Through the use of threads, the model is trained, evaluated, and tested across different datasets, significantly reducing the time required for evaluation. To ensure compatibility with the same model, alphabetic sequences are converted into their corresponding ASCII representation.

In refining accuracy, we conducted fine-tuning by experimenting with various parameter values. It was discovered that altering the number of inputs had the most significant impact. These enhancements collectively aim to improve the functionality and effectiveness of the multi-sequence learning model, broadening its applicability across diverse domains.

II. LITERATURE SURVEY

A. Exploring the Neocortex and its Computational Model: Hierarchical Temporal Memory (HTM)

The neocortex is a highly evolved region of the brain found in mammals, particularly prominent in humans. It plays a crucial role in sensory perception, motor commands, spatial reasoning, conscious thought, and language. Structurally, the neocortex is organized into layers, with different regions specializing in various cognitive functions as shown in Figure 1.

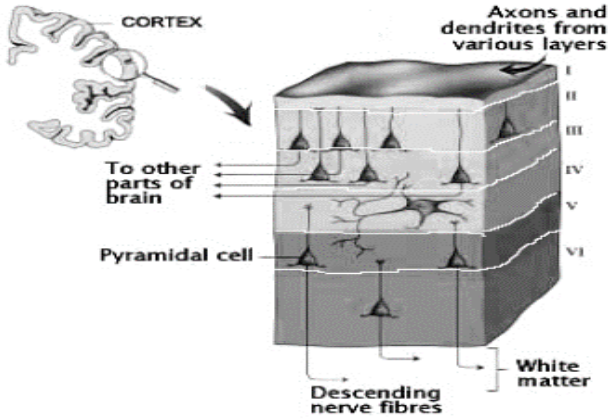

Figure 1: Neocortex Layers

Hierarchical Temporal Memory (HTM) is a computational model of how the neocortex processes information. It is based on the idea that the neocortex learns and recognizes patterns in sensory input through a hierarchy of regions, each processing information at different levels of abstraction and timescales.

In the HTM model:

Spatial Pooling: This stage corresponds to the sensory input received by the neocortex. Neurons in the first layer (akin to the input layer in neural networks) detect patterns in the input and activate accordingly. This process is similar to how sensory information is received by the neocortex from various sensory organs.

Temporal Memory: This stage involves learning temporal sequences and patterns over time. Neurons in subsequent layers form connections based on the temporal relationships between patterns. This aspect of the model mimics the neocortex's ability to recognize sequences of patterns and make predictions based on them. It is crucial for tasks such as predicting future events based on past experiences.

Hierarchy: The HTM model incorporates the concept of a hierarchical structure similar to that of the neocortex. Higher-level regions in the hierarchy learn more complex patterns and representations by integrating information from lower-level regions. This hierarchical organization enables the neocortex to process information at varying levels of abstraction, from simple sensory features to complex concepts.

Feedback Connections: Feedback connections in the HTM model allow for context and feedback mechanisms, which are essential for refining predictions and updating representations based on new information. This aspect mirrors the feedback loops present in the neocortex, enabling it to continually refine and update its understanding of the environment.

Overall, the HTM model is inspired by the structural and functional organization of the neocortex, aiming to replicate its capabilities in processing sensory information, learning temporal sequences, and forming hierarchical representations. A pictorial representation of both biological and HTM neuron is shown below in Figure 2. It provides a framework for understanding how the neocortex might perform these tasks computationally, offering insights into both the biological basis of cognition and potential applications in artificial intelligence.

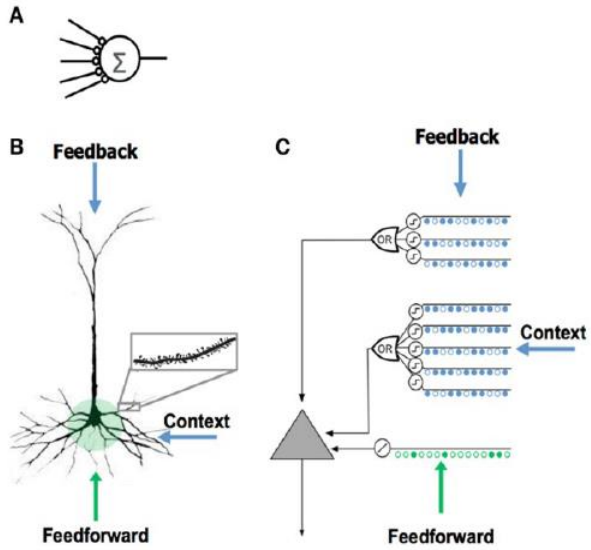

Figure 2: Comparison between Biological and HTM neuron

B. SDRs

SDRs, consisting of thousands of bits where only a small portion are active at any given time, encode semantic attributes without explicit labelling. This allows for automatic generalization based on semantic similarity, enabling flexible and creative intelligence. HTM theory elaborates on creating, storing, and recalling SDRs and sequences, highlighting the dynamic nature of neural representations and associative linking between different populations of neurons.

In contrast to traditional computer data structures, brains employ Sparse Distributed Representations (SDRs) for knowledge representation, facilitating flexible and nuanced understanding of concepts. SDRs consist of active and inactive bits, with each bit encoding semantic attributes, allowing for automatic generalization based on similarity between representations [1].

C. Spatial Pooling

The Spatial Pooler is a crucial component of the Hierarchical Temporal Memory (HTM) model, emulating the neocortex's ability to recognize and learn spatial patterns in sensory input. It operates by receiving binary or scalar input representing features from various sensory modalities and organizing them into a grid of cells arranged in columns as shown in Figure 3. These cells have associated connection weights with input bits, determining their influence on cell activation. Through overlap calculation between input patterns and cell connections, a subset of cells with the highest overlaps is selected as winners, generating a sparse distributed representation (SDR) of the input pattern. The Spatial Pooler dynamically adjusts its connections based on input patterns, strengthening connections to active bits while weakening connections to inactive ones, facilitating efficient representation of relevant features. This pre-processing step transforms raw sensory input into robust SDRs suitable for further processing by the HTM model's Temporal Memory and higher-level regions in the hierarchy [2].

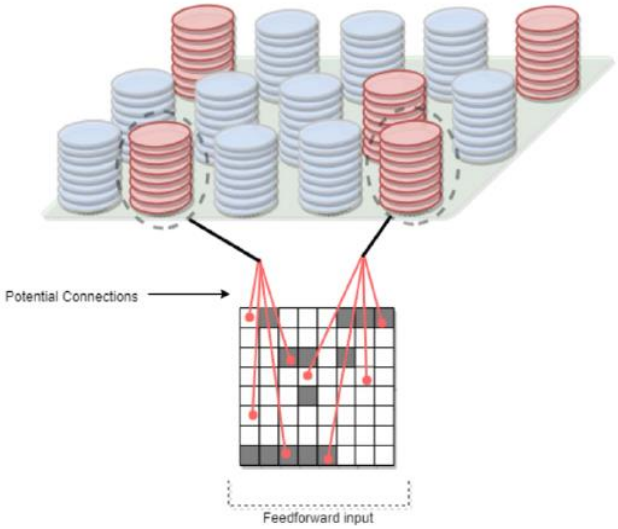

Figure 3 : Column in spatial pooling and potential connections.

- Proximal Dendrite Segments: The relationship between spatial pooling and proximal and distal dendrite segments lies in how input patterns are processed and integrated within the HTM framework. Proximal dendrite segments primarily receive input from neighbouring columns or proximal inputs, representing the spatial context of the input data. These inputs contribute to the activation of columns during spatial pooling, influencing the selection of winning columns and the creation of sparse representations. The Figure 4 below shows the pictorial representation of proximal dendrite segment.

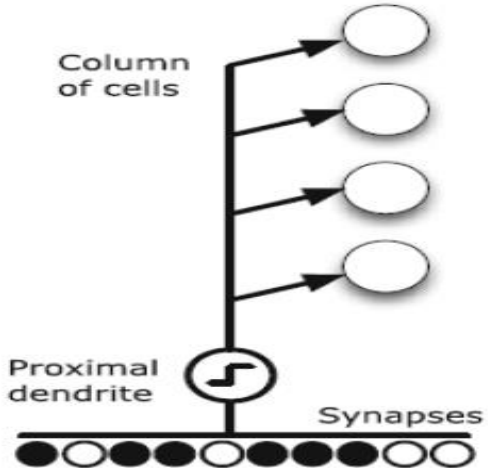

Figure 4 : Proximal Dendrite Representation

- Distal Dendrite Segments: Distal dendrite segments play a role in capturing temporal sequences and contextual information. They receive inputs from more distant columns, representing the temporal context or associations between different input patterns over time. The connections formed by distal dendrite segments allow columns to learn predictive relationships between sequences of input patterns, enabling the HTM model to make predictions based on learned temporal patterns.

D. Hierarchical Temporal Memory (HTM)

Temporal Memory, a core component of Hierarchical Temporal Memory (HTM), mirrors the neocortex's capacity to process time-based patterns. It learns temporal associations between sparse distributed representations (SDRs) from the Spatial Pooler, aiding in prediction and inference based on past patterns. Organized into columns, cells within Temporal Memory denote active, predictive, or inactive states, facilitating sequence formation [3]. Through temporal pooling, it reinforces connections between cells representing consistent sequences, enabling accurate predictions. Predictive inference anticipates future patterns by activating cells in predictive states, while continuous learning and adaptation adjust connections based on pattern consistency. Temporal Memory is pivotal in HTM, supporting the learning of temporal sequences, predictive inference, and stable pattern representation.

III. METHODOLOGY

The project "Enhanced Multi-Sequence Learning with Improved Accuracy" is developed using C# .Net Core within the Microsoft Visual Studio 2022 Integrated Development Environment (IDE). This project utilizes an open-source implementation of Hierarchical Temporal Memory (HTM) in C#/.Net Core to explore the functionality of the HTM model while it learns sequences of integers and alphabets, with the objective of enhancing the model's accuracy. Unlike the previous implementation of the Multi-Sequence Learning project, which employed a small dataset for training, our goal is to not only enhance the accuracy of the model but also to expand the size of the dataset used.

A. Datasets

This section describes the data set used for the project.

i. Sequence of Alphabets: Random sequence of alphabets in increasing order.

Table 1 : Date Sets (Alphabets)

Label	Sequence
S1	BDFGJKLORSUV
S2	DGJKLMNPQVXY
S3	CFHIKNSTUWY
S4	ACHIJLSVXYZ
S5	EFKLQSUWZ

Table 1 sequences are sampled from a data set randomly generated and the size of each sequence is between 10-20 characters long.

ii. Sequence of Numbers: Random sequence of numbers in increasing order.

Table 2 : Date Sets (Numbers)

Label	Sequence
S1	1,3,7,10,11,13,14,15,18,26,31
S2	1,2,6,9,11,12,13,19,28,29,34,36,38,42,49
S3	2,16,19,25,32,33,34,37,40,47
S4	5,8,11,17,19,21,22,23,24,32,36,48
S5	7,10,13,16,20,24,26,29,32,34,36,39,42,43

Table 2 sequences are sampled from a data set randomly generated and the size of each sequence is between 10-20 numbers long.

B. Encoders

In Hierarchical Temporal Memory (HTM), an encoder is a pivotal component tasked with converting raw sensory input data into Sparse Distributed Representations (SDRs), the primary language for information representation within HTM systems. Encoders play a critical role in transforming diverse types of input data, including scalar values, temporal data, categorical variables, and spatial coordinates, into high-dimensional binary vectors while preserving relevant information and semantic relationships [4]. Once the input data is encoded into SDRs, it can be fed into HTM networks for further processing. By mapping input data into a common sparse format, encoders facilitate efficient processing and

learning within HTM networks, enabling the modelling of complex temporal and hierarchical relationships across various domains.
For encoding alphabets and number sequences, scalar encoders are used, that are utilized for encoding scalar (single dimension encoded data). They partition the input space into multiple subintervals and represent each value with a sparse binary vector. Table 3 and Table 4 shows encoder parameters for alphabets and numbers respectively.

Table 3 : Encoder for alphabets

Parameter	Values
Maximum value	122
Minimum value	97
Resolution(W)	15
Radius	-1
Periodicity	False
Clip input	False
Size of output SDR(N)	100

Table 4 : Encoder for numbers

Parameter	Values
Maximum value	50
Minimum value	0
Resolution(W)	15
Radius	-1
Periodicity	False
Clip input	False
Size of output SDR(N)	100

In the given setup, the encoder's range for alphabet characters spans from 97 to 122 inclusively. This range is determined by the ASCII values assigned to the lowercase letters, where 'a' corresponds to 97 and 'z' to 122. Consequently, any input values representing alphabetic characters fall within this specified range.
Conversely, the encoder's range for numerical values falls between 0 and 50. This range is set to encompass values from 0 to 50 inclusively.

C. HTM Configuration

In the context of learning random alphabet sequences within a Hierarchical Temporal Memory (HTM) network, several key configuration values play crucial roles in shaping the network's behavior and learning capabilities. It was observed that altering the number of input bits had a more significant impact on the accuracy. The numColumns parameter determines the spatial extent of the network, while CellsPerColumn influences the granularity of representation within each column. In the context of learning random alphabet sequences, setting global inhibition to true may result in more robust pattern separation and increased selectivity in column activations. A moderate value for LocalAreaDensity can ensure sufficient sparsity in column activations. It is crucial to adjust the PotentialRadius to cover a reasonable range of inputs to capture potential temporal patterns. Parameters such as MaxBoost and InhibitionRadius should be set to moderate values to promote stable learning dynamics and competition between columns. MaxSynapsesPerSegment can be adjusted to accommodate the complexity of the alphabet sequences, while ActivationThreshold and ConnectedPermanence should be set to appropriate levels to ensure sensitivity to input patterns and robust connectivity. The values for PermanenceDecrement and PermanenceIncrement should allow for gradual adaptation and learning. Lastly, PredictedSegmentDecrement can be adjusted to reinforce predictive behaviors effectively. Overall, a balanced configuration with moderate parameter values would facilitate the learning of random alphabet sequences within an HTM network, allowing for effective pattern recognition and prediction. Table 5 shows HTM configurations that are used.

Table 5 : HTM Configuration parameters

Parameters	Default value
inputBits	200
numColumns	1024
CellsPerColumn	25
GlobalInhibition	true
LocalAreaDensity	-1
NumActiveColumnsPerInhArea	0.02 * numColumns
PotentialRadius	0.15 * inputBits
MaxBoost	10
InhibitionRadius	15
DutyCyclePeriod	25
MinPctOverlapDutyCycles	0.75
MaxSynapsesPerSegment	0.02 * numColumns
ActivationThreshold	15
ConnectedPermanence	0.5
PermanenceDecrement	0.25
PermanenceIncrement	0.15
PredictedSegmentDecrement	0.1

IV. IMPLEMENTATION

This section demonstrates the enhancements made to the existing work on multi-sequence learning for numbers and alphabets. While the prior research provided valuable insights into the workings of the HTM algorithm, its application was constrained by a small dataset used for model training. Moreover, the emphasis was primarily on understanding the learning process of numbers, alphabets, and images, with limited discussion on testing accuracy.

In this implementation, we aimed to address these limitations by augmenting the dataset with a greater number of sequences and incorporating separate evaluation and testing datasets to rigorously evaluate the model's accuracy. Our implementation is structured into three main phases. Firstly, we generate new training, evaluation, and testing datasets based on user input. Secondly, we preprocess the training dataset into a format suitable for training our model. Lastly, we train the HTM model using the prepared training dataset and assess its accuracy using the evaluation and testing datasets.

A. Preparation of Dataset

To initiate the learning process as shown in Figure 5, the creation of suitable datasets stands as a pivotal initial step. In pursuit of this objective, a robust sequence generator tailored to accommodate both alphanumeric sequences with ease. This sophisticated tool is designed to operate seamlessly,

guided by user-specified parameters that dictate the generation process. Among these parameters, users can define the number of sequences required and set the range of values within each sequence, ensuring flexibility and customization. By specifying maximum and minimum values, users can precisely tailor the characteristics of the datasets to suit their specific needs.

Moreover, recognizing the importance of robust evaluation and testing, the generator goes a step further by automatically generating supplementary datasets. These additional datasets serve a crucial purpose in assessing the accuracy and performance of the model. By randomly selecting subsequences from the primary dataset, the evaluation and test datasets provide a diverse range of scenarios for rigorous testing. This comprehensive approach ensures that the model's capabilities are thoroughly scrutinized, leading to refined performance and enhanced reliability.

While the HTM model demonstrates proficiency in handling numeric sequences, accommodating alphabetic sequences necessitates a transformative approach. To address this challenge, a novel method harnessing ASCII encodings is used. This innovative technique enables the seamless integration of alphabetic data into the HTM model's framework. Initially, alphabet sequences undergo a transformation process wherein they are converted into their corresponding ASCII representations. These representations, imbued with the essence of the original data, are then seamlessly integrated into the learning process of the HTM model. This integration not only expands the model's capabilities but also enriches its ability to handle diverse data types with precision and efficiency.

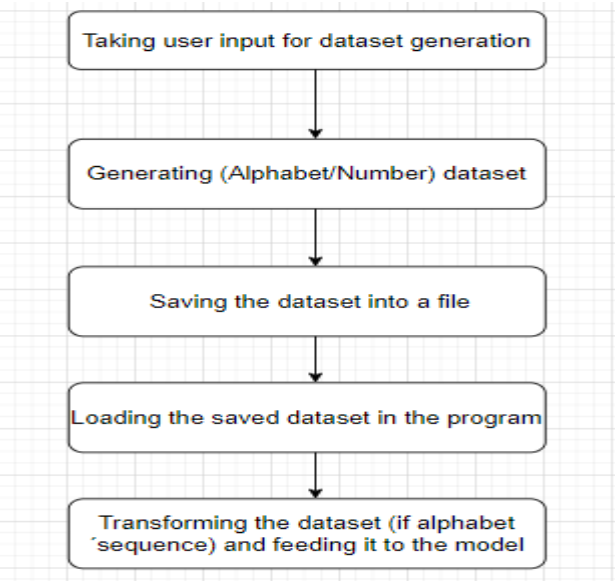

Figure 5 : Generating and preparing dataset

.

B. Learning Phase

During the training phase, the first step is loading the data that is done by the process described above. This data is fed into an encoder. The encoder then extracts essential features, normalizer the data and most importantly transform the input into a sparse distributed representation (SDR). The output of encoder is then passed to the spatial pooler which is initialized to the default parameters specified in the code file. It receives the encoded input and computes a set of active columns, creating a sparse representation where only a subset of columns is active at any given time. It is trained for every input over several iterations and this continues until a stable state in reached. The homeostatic plasticity controller maintains a balance between stability and adaptability. It adjusts the strength of synaptic connections based on network activity levels, preventing overfitting and ensuring the HTM remains capable of learning new patterns while retaining previously acquired knowledge. Once the Sparse Distributed Representation (SDR) for each input becomes stable, the Homeostatic Plasticity Controller (HPC) initiates an event signifying the achievement of a stable state by the Spatial Pooler [5]. The time taken by the spatial pooler to achieve a stable state depends on the range of input data (alphabets/number). Figure 6, illustrates the learning phase of dataset.

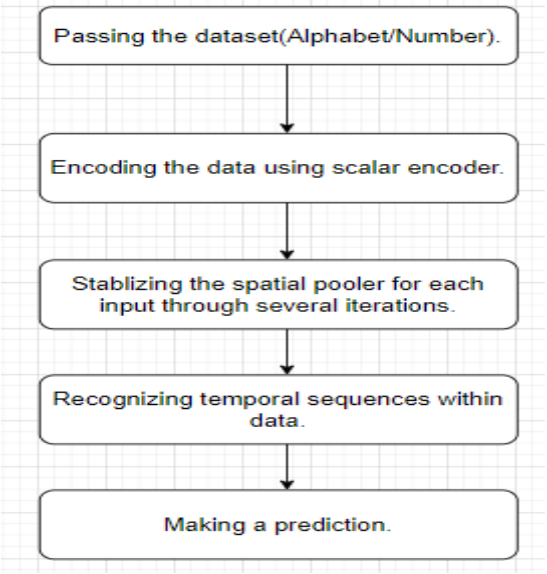

Figure 6 : Learning process dataset

.

C. Prediction

After undergoing the learning process, the Hierarchical Temporal Memory (HTM) model utilizes its temporal memory component to make predictions. Initially, raw sensory input is preprocessed by the encoder, extracting essential features and preparing the data for HTM processing. This encoded input then enters the spatial pooler, which forms sparse distributed representations (SDRs) by activating a subset of columns based on input patterns. Subsequently, the SDR is fed into the temporal memory, where temporal sequences are learned by establishing connections between active columns over time. This temporal memory maintains a representation of the current state, derived from the sequence of past inputs. Leveraging this learned temporal context and the current input, the temporal memory predicts future inputs by activating columns representing potential next states based on the recognized patterns and current context.

To be more specific, after learning the sequences, the algorithm creates a predictor tasked with predicting the subsequent element in a sequence. The evaluation or test dataset is then provided to the predictor, which attempts to predict the next element in the sequence for each element within it. While the predictor may offer multiple predictions, each one is paired with a similarity score indicating the confidence level of the prediction. Therefore, if multiple elements are predicted, we select the prediction with the highest similarity score. Once a sequence has been predicted, its accuracy is determined by dividing the number of correctly predicted subsequent elements by the total predictions, as follows:

Correctly predicted elements / total predictions * 100

V. RESULTS

Our project is aimed at improving a multi-sequence learning program. We have implemented several enhancements to achieve this goal. Firstly, we have introduced the capability to generate random number and alphabet datasets based on user preferences. Additionally, we have enabled the program to handle multiple sequences concurrently, allowing for simultaneous learning, evaluation, and testing. Moreover, efforts have been made to enhance the accuracy of the program that included fine tuning different parameters of the HTM configuration. It was seen that increasing the number of input bits in the Hierarchical Temporal Memory (HTM) model to 200 resulted in improved test accuracy due to several factors. This enhancement offered the model increased representation capacity, allowing it to capture finer details and complexities in the input data. The larger number of bits also improved discriminative power, enabling the model to differentiate between similar sequences more effectively and reduce ambiguity. Additionally, reduced information loss during encoding preserved more detailed information, leading to richer representations. This, coupled with improved generalization capabilities, enabled the model to make more accurate predictions, even with unseen or slightly different sequences in the test dataset. Overall, the increase in the number of bits enhanced the model's performance by providing greater expressive power, improved representation quality, and better generalization abilities.

Figure 7 shows the simplified illustration of how multi sequence learning experiment works: When the program starts, it asks the user whether they want to work with numbers or letters. Then, it prompts the user to specify how many datasets they want to work with simultaneously. Depending on the user's choices, the program either generates new datasets or uses existing ones. If the user selects alphabet sequences, the program converts them into ASCII representation before beginning the learning process.

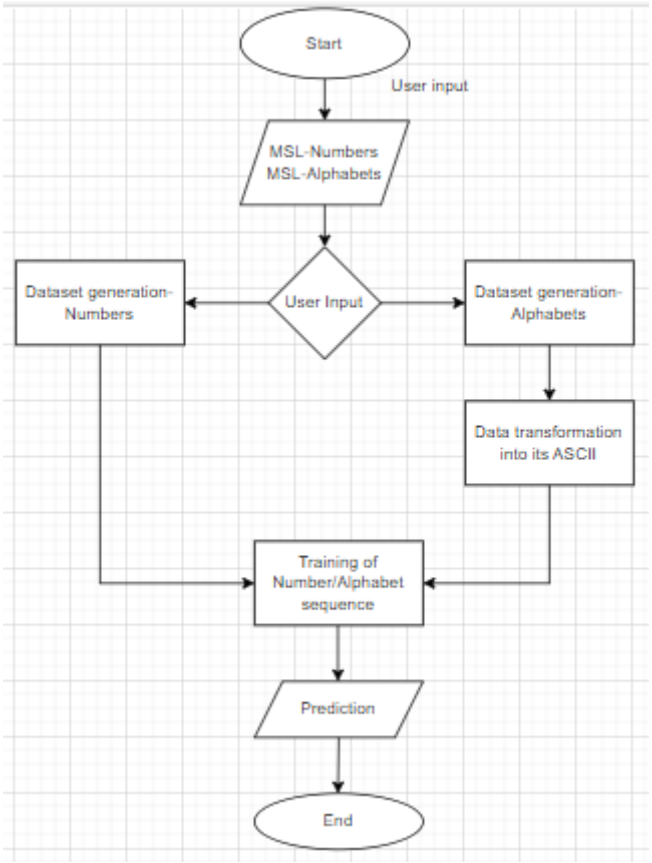

Figure 7 : Flow chart for Enhance Multi Sequence Learning Experiment

A. Multi Sequence Learning – Sequence of Numbers

Figure 8 and Figure 9 below illustrate the model in the process of predicting sequences within the evaluation and test datasets. Following training, the model demonstrates the ability to predict each sequence with an accuracy ranging from 35% to 100%, averaging at 53%. This marks a significant improvement compared to the previous implementation, where the test accuracy for number datasets was below 25%.

```
Predicted Sequence: S6_46 - Predicted next element: 25
Input: 22
Predicted Sequence: S8_33 - Predicted next element: 24
Input: 24
Predicted Sequence: S8_37 - Predicted next element: 33
Input: 33
Predicted Sequence: S8_37 - Predicted next element: 37
Input: 37
Nothing predicted
Input: 37
Predicted Sequence: S10_ - Predicted next element: 47
Input: 38
Predicted Sequence: S4_41 - Predicted next element: 38
Accuracy for E8 sequence: 57.142857142857142%
Input: 9
Predicted Sequence: S6_10 - Predicted next element: 10
Input: 10
Predicted Sequence: S9_26 - Predicted next element: 15
Input: 15
Predicted Sequence: S9_28 - Predicted next element: 26
Input: 26
Predicted Sequence: S9_29 - Predicted next element: 28
Input: 28
Predicted Sequence: S9_36 - Predicted next element: 29
Input: 29
Predicted Sequence: S9_37 - Predicted next element: 36
Input: 36
Predicted Sequence: S9_47 - Predicted next element: 37
Accuracy for E9 sequence: 100%
Input: 28
Predicted Sequence: S9_36 - Predicted next element: 29
Input: 35
Predicted Sequence: S10_47 - Predicted next element: 37
Input: 37
Predicted Sequence: S10_48 - Predicted next element: 47
Input: 47
Predicted Sequence: S10_49 - Predicted next element: 48
Accuracy for E10 sequence: 75%
```

Figure 8 :Accuracy of evaluation dataset– Sequence of Numbers

```
Predicted Sequence: S7_15 - Predicted next element: 15
Input: 15
Nothing predicted
Input: 15
Predicted Sequence: S9_28 - Predicted next element: 26
Input: 16
Predicted Sequence: S5_21 - Predicted next element: 19
Input: 17
Predicted Sequence: S7_26 - Predicted next element: 19
Accuracy for T7 sequence: 33.33333333333333%
Input: 24
Predicted Sequence: S8_37 - Predicted next element: 33
Input: 33
Predicted Sequence: S8_37 - Predicted next element: 37
Input: 37
Nothing predicted
Input: 37
Predicted Sequence: S10_ - Predicted next element: 47
Input: 38
Predicted Sequence: S4_41 - Predicted next element: 38
Accuracy for T8 sequence: 60%
Input: 9
Predicted Sequence: S6_10 - Predicted next element: 10
Input: 10
Predicted Sequence: S9_26 - Predicted next element: 15
Input: 15
Predicted Sequence: S9_28 - Predicted next element: 26
Input: 26
Predicted Sequence: S9_29 - Predicted next element: 28
Accuracy for T9 sequence: 100%
Input: 27
Predicted Sequence: S10_28 - Predicted next element: 27
Input: 28
Predicted Sequence: S10_37 - Predicted next element: 35
Input: 35
Predicted Sequence: S10_47 - Predicted next element: 37
Input: 37
Predicted Sequence: S10_48 - Predicted next element: 47
Input: 47
Predicted Sequence: S10_49 - Predicted next element: 48
Accuracy for T10 sequence: 80%
```

Figure 9 : Accuracy of test dataset– Sequence of Numbers

B. Multi Sequence Learning – Sequence of Alphabets

Figure 10 and Figure 11 below illustrate the model in the process of predicting sequences within the evaluation and test datasets. Following training, the model demonstrates the ability to predict each sequence with an accuracy ranging from 35% to 100%, averaging at 53%. This marks a significant improvement compared to the previous implementation, where the test accuracy for number datasets was below 25%.

```
Input: 108
Predicted Sequence: S1_ - Predicted next element: 120
Input: 110
Predicted Sequence: S8_113 - Predicted next element: 111
Input: 111
Predicted Sequence: S10_113 - Predicted next element: 112
Input: 113
Predicted Sequence: S8_116 - Predicted next element: 114
Accuracy for E8 sequence: 66.66666666666666%
Input: 102
Predicted Sequence: S2_106 - Predicted next element: 103
Input: 103
Predicted Sequence: S9_105 - Predicted next element: 104
Input: 104
Predicted Sequence: S9_107 - Predicted next element: 105
Input: 105
Predicted Sequence: S9_108 - Predicted next element: 107
Input: 107
Predicted Sequence: S9_114 - Predicted next element: 108
Input: 108
Predicted Sequence: S9_115 - Predicted next element: 114
Accuracy for E9 sequence: 100%
Input: 97
Predicted Sequence: S10_100 - Predicted next element: 98
Input: 98
Predicted Sequence: S10_104 - Predicted next element: 100
Input: 100
Predicted Sequence: S10_110 - Predicted next element: 104
Input: 104
Predicted Sequence: S10_111 - Predicted next element: 110
Input: 110
Predicted Sequence: S10_112 - Predicted next element: 111
Input: 111
Predicted Sequence: S10_113 - Predicted next element: 112
Input: 112
Predicted Sequence: S10_114 - Predicted next element: 113
Input: 113
Predicted Sequence: S10_116 - Predicted next element: 114
Input: 114
Predicted Sequence: S10_118 - Predicted next element: 116
Accuracy for E10 sequence: 100%
```

Figure 10 : Accuracy of evaluation dataset– Sequence of Alphabets

```
Input: 111
Predicted Sequence: S10_113 - Predicted next element: 112
Input: 113
Predicted Sequence: S8_116 - Predicted next element: 114
Input: 114
Predicted Sequence: S10_118 - Predicted next element: 116
Input: 116
Predicted Sequence: S10_119 - Predicted next element: 118
Accuracy for T8 sequence: 60%
Input: 103
Predicted Sequence: S4_108 - Predicted next element: 104
Input: 104
Predicted Sequence: S9_107 - Predicted next element: 105
Input: 105
Predicted Sequence: S9_108 - Predicted next element: 107
Input: 107
Predicted Sequence: S9_114 - Predicted next element: 108
Input: 108
Predicted Sequence: S9_115 - Predicted next element: 114
Input: 114
Predicted Sequence: S9_118 - Predicted next element: 115
Accuracy for T9 sequence: 100%
Input: 98
Predicted Sequence: S5_102 - Predicted next element: 101
Input: 100
Predicted Sequence: S10_110 - Predicted next element: 104
Input: 104
Predicted Sequence: S10_111 - Predicted next element: 110
Input: 110
Predicted Sequence: S10_112 - Predicted next element: 111
Input: 111
Predicted Sequence: S10_113 - Predicted next element: 112
Input: 112
Predicted Sequence: S10_114 - Predicted next element: 113
Input: 113
Predicted Sequence: S10_116 - Predicted next element: 114
Input: 114
Predicted Sequence: S10_118 - Predicted next element: 116
Input: 116
Predicted Sequence: S10_119 - Predicted next element: 118
Accuracy for T10 sequence: 88.88888888888889%
```

Figure 11 : Accuracy of test dataset– Sequence of Alphabet

Unit Testing:

Data Generator: Figure 12 below illustrates the successful test cases for various methods of data set generation. These cases verify whether the generated data is in the correct order and if it complies with the specified parameters. Additionally, they examine edge cases by providing invalid parameters and checking if the functions produce errors.

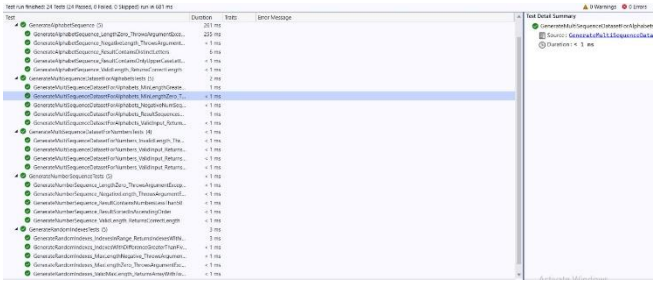

Figure 12 : Unit test for Data generator

Helper Method: Figure 13 below displays successful test cases for various helper functions. These functions encompass methods for ASCII conversion, extracting a sub-array from a larger array, and generating filenames in the correct format. These test cases evaluate the accuracy of ASCII conversion, ensure the removal of non-alphabetic characters, verify that the extracted sub-array is contained within the larger array, and validate the formatting of filenames. Additionally, these tests cover edge cases by providing invalid parameters and checking if the functions appropriately handle errors by throwing exceptions.

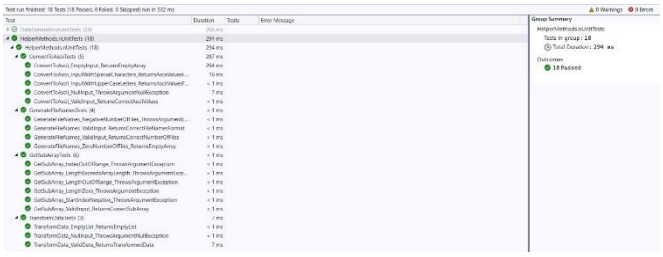

Figure 13 : Unit test for Helper Method

VI. CONCLUSION

This project was dedicated to enhancing the multi-sequence learning model comprehensively. Various strategies were employed to achieve this goal, including the development of a dataset generator to facilitate the creation of new datasets for training, evaluation, and testing. Additionally, the model was evaluated on different datasets by utilizing threads for enhanced efficiency. Another crucial aspect was enhancing accuracy, which involved exploring different parameters and adjusting values that had a significant impact.

While considerable effort has been dedicated to enhancing the previous implementation of the multi-sequence learning model, this project remains imperfect. There are several aspects that could benefit from further improvement. One area for enhancement involves refining test accuracy through the utilization of alternative techniques. Additionally, improvements related to the dataset are warranted. It has been observed that when presenting a sequence lacking any discernible order (such as ascending or descending) to the model, it exhibits signs of catastrophic forgetting, leading to the erasure of previously learned sequences. Addressing these issues presents opportunities for improvement.

VII. References

[1] "Sparse Distributed Representations Sparse Distributed Representations,"2016. Available:https://www.numenta.com/assets/pdf/biological-and-machine-intelligence/BaMI-SDR.pdf

[2] Y. Cui, S. Ahmad, and J. Hawkins, "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding," Frontiers in Computational Neuroscience, vol. 11, Nov. 2017, doi: https://doi.org/10.3389/fncom.2017.00111.

[3] Kjell Jørgen Hole, "The HTM Learning Algorithm," Springer eBooks, pp. 113–124, Jan. 2016, doi: https://doi.org/10.1007/978-3-319-30070-2_11.

[4] S. Purdy, "Encoding Data for HTM Systems.," arXiv (Cornell University), Feb. 2016.

[5] D. Dobric, A. Pech, B. Ghita, and T. Wennekers, "Improved HTM Spatial Pooler with Homeostatic Plasticity Control," Proceedings of the 10th International Conference on Pattern Recognition

Applications and Methods, 2021, doi: https://doi.org/10.5220/0010314200980106.