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

Multi Sequence Learning with Language Semantic

Chaitra Shrikanth Bhandari *chaitra.bhandari@stud.fra-uas.de*
Dharani Kumthi *dharani.kumthi@stud.fra-uas.de*
Gosi Bhavani *gosi.gosi-bhavani@stud.fra-uas.de*

***Abstract*—This paper proposes an extension to a Multi-Sequence Learning experiment by incorporating language semantics. The existing system excels at learning and predicting multi sequence data. This enhancement introduces the ability to process textual sequences and generate follow-up text based on the learned patterns. To implement this methodology, an automated system is developed to load text data, train the model, and output accuracy metrics. The model is tested with test data and user input to analyze the prediction accuracy. The results of the study demonstrate the effectiveness of prediction accuracy in evaluating sequence generation. By calculating the percentage of correctly predicted next elements out of the total number of predictions, researchers can obtain valuable insights into the performance of text generation models. Additionally, the developed system offers a mechanism for user input for generating text sequences and assessing binary cross-entropy loss during the training phase, thereby facilitating further research in the field of natural language processing and text generation.**

***Index Terms*—Spatial Pooler(SP), Sparse Distributed Representations(SDR), Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA), SynapticPlasticity, NeoCortex, Spatial Pooler(SP), Temporal memory(TM), Binary Cross Entropy(BCE), Multi Sequence Learning(MSL).**

I. Introduction

The Hierarchical Temporal Memory (HTM) algorithm is employed in Multisequence Learning for training neural networks to learn sequences. The Neocortex package integrates the Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA) within the C#/.Net Core environment. It encompasses essential components such as the Spatial Pooler, Temporal Memory, various encoders, and Cortical Network algorithms. [7]

The paper utilizes the training of a neural network to learn sequences using the Hierarchical Temporal Memory (HTM) algorithm. Utilizing HTM for multi-sequence learning proves to be a potent approach for recognizing and predicting patterns across diverse input sequences.

This paper aims to implement methods, along with Multisequence Learning, with the goal of converting text data into ASCII and using it as training input for the model. It will then calculate binary cross-entropy loss during training and evaluate prediction accuracy upon completion.By leveraging established multi-sequence learning techniques, our primary focus is on generating predictive code from text data, specifically emphasizing high accuracy in prediction and developing inference code in C#/.NET Core.

II. Literature Survey

A. HTM (Hierarchical Temporal Memory)

Hierarchical Temporal Memory (HTM) emerged in 2005 as a groundbreaking machine learning paradigm proposed by Jeff Hawkins, drawing inspiration from the intricate workings of the human brain. The foundation of HTM lies in its attempt to replicate the cognitive processes of the neocortex, the seat of higher-order thinking in mammals. By mirroring the brain's hierarchical structure and memory-prediction framework, HTM aims to unlock the potential for machines to comprehend and predict complex patterns.[4][5]

Biological Inspiration: At its core, HTM is deeply rooted in neuroscience, particularly the structural and algorithmic properties of the neocortex. Hawkins and his team sought to emulate the brain's ability to process sensory information, recognize patterns, and anticipate future events. By embracing the biological principles of cortical computation, HTM introduces a fresh perspective to artificial intelligence, paving the way for more biologically plausible learning algorithms.[6][9]

HTM networks are characterized by their hierarchical organization, which allows for the abstraction of information across multiple levels of granularity. Within this framework, HTM employs sparse distributed representations (SDRs) to encode data efficiently, enabling the simultaneous representation of distinct items with minimal interference. [10]These representations, combined with realistic neuron models equipped with active dendrites and thousands of synapses, empower HTM networks to learn time-based sequences and adapt to dynamic environments in real-time.Over the years, HTM has witnessed significant advancements and garnered attention for

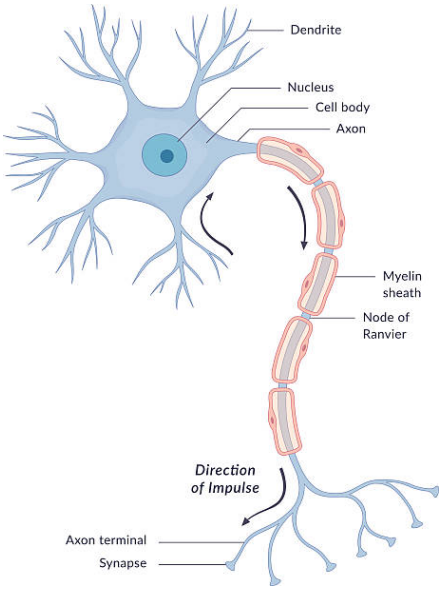

Figure 1. Nerve Cell

its remarkable capabilities in anomaly detection and sequence prediction tasks. Its prowess in processing streaming data has propelled it to the forefront of research, with HTM networks achieving state-of-the-art performance in various domains. As ongoing research endeavors continue to refine HTM algorithms and broaden their applicability, the future holds immense promise for this biologically-inspired approach to machine learning.[2][8]

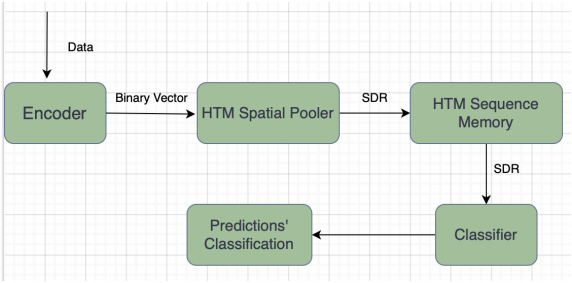

Figure 2. HTM Pipeline

B. Sparse Distributed representations (SDRs)

SDRs are a type of encoding used in the brain's neocortex to represent information. In SDRs, only a small percentage of neurons are active at any given time, while the majority remain inactive. This sparsity enables efficient representation of information while conserving neural resources.[14]

1) Distributed Encoding:

Information in SDRs is encoded not by the activity of individual neurons, but rather by the collective activity of a group of neurons. This distributed encoding means that each bit of information is represented by the pattern of activation across a population of neurons, rather than by the activity of a single neuron. This distributed nature allows for robustness and fault tolerance in representing and processing information.

2) Versatile and Scalable:

SDRs are versatile and can encode a wide range of information, from sensory features to abstract concepts. They are also scalable, meaning they can represent complex information patterns across various scales without significantly increasing the number of active neurons. This scalability makes SDRs suitable for modeling the hierarchical organization of the neocortex and for applications in machine learning and artificial intelligence.

C. Spatial Pooler

1) Role of the Spatial Pooler (SP):

The SP is a crucial component of HTM networks, responsible for converting input patterns into Sparse Distributed Representations (SDRs) in real-time.It operates continuously, processing incoming data streams and transforming them into sparse representations that capture essential features while conserving neural resources.[3]

2) Structure and Function of Mini-columns:

Within HTM networks, a layer consists of mini-columns, each containing cells that mimic the dendrite properties of pyramidal cells in the neocortex.

3) Local Inhibition Mechanism:

The SP implements local inhibition among neighboring mini-columns, enabling a k-winners-take-all computation.Only a small fraction of mini-columns with the most active inputs become active, ensuring sparsity and promoting efficient representation of information.[10]

4) SynapticPlasticity and Boosting Mechanism:

Synaptic connections onto active cells are modified based on Hebbian learning rules, reinforcing active input connections and punishing inactive ones.A homeostatic excitatory control mechanism known as "boosting" adjusts the relative excitability of mini-columns, encouraging inactive ones to become active and participate in representing the input.[9][11]

These components collectively enable the SP to efficiently encode input patterns into sparse representations, facilitating further processing and learning within the HTM network.

D. SDR Encoder and Classifier

The HTM sequence memory operates with sparse distributed representations (SDRs) internally. To apply HTM to real-world sequence learning problems, we need to first convert the original data to SDRs using an encoder. The encoder translates real valued inputs into SDRs that are passed through the spatial pooler and temporal memory modules, where they cause state changes in cells and predictions of

what inputs will be seen next. The state of the cells is then streamed out to a decoder module, which translates these sparse activations of cells back into real valued outputs, to interface with other systems.
The HTM classifier utilizes a hierarchical structure composed of layers of mini-columns and cells, mimicking the organization of the neocortex. This architecture enables efficient processing of input data and facilitates learning of complex patterns.Incorporating temporal memory, the HTM classifier can learn and predict sequences of input patterns. Temporal memory cells within the network learn temporal dependencies, allowing the classifier to make accurate predictions based on previous inputs.

E. Temporal Memory

The functioning of Temporal Memory involves preserving a collection of active cells that embody the present context of the input data. Upon receiving fresh input patterns, the active cells undergo modifications according to the similarity between the input and the current context. Additionally, the algorithm manages a set of connections between cells that represent the sequence of patterns that were encountered previously.[12] Using these connections, the Temporal Memory can predict the next likely input pattern based on the current context. If the prediction is correct, the algorithm reinforces the connections between the cells that were active during the predicted sequence. If the prediction is incorrect, the algorithm adjusts the connections to reduce the likelihood of that sequence occurring in the future.[2]

F. Multisequence Learning

Multisequence Learning deals with learning patterns across multiple sequences of data. The focus is on modeling the dependencies and interactions between elements within and potentially between different sequences. It's a broader concept encompassing tasks like natural language processing, which requires the analysis of multiple sentences, or time series prediction based on past observations in a sequence. Given its versatility, Multisequence Learning has applications in various fields where understanding and extracting patterns from sequential data are essential.

G. Predictor

This paper discusses the 'Predictor' class, a key component for sequence learning tasks. The 'Run' method of the 'MultiSequenceLearning' class outputs an instance of this class after the learning process is complete. The 'Predictor' class allows for predicting the next element in a sequence. It takes a list of input elements as input and attempts to forecast the following element for each one. The accuracy of the predictions generally improves as the length of the sequence presented increases.
The 'Predictor' object returned by the 'Run' method is actually an instance of a specific subclass that inherits from the abstract class 'PredictorBase' (defined in the NeoCortexApi library). This subclass determines the specific prediction method based on the training data.The abstract 'PredictorBase' class provides a standardized interface for making predictions across various tasks like classification, anomaly detection, and sequence prediction. It defines a 'Compute' method that takes an input vector and returns a list of predicted values. The specific implementation details of 'Compute' vary depending on the chosen subclass of 'PredictorBase'.
The Predictor object generated by the 'Run()' method encapsulates the internal state of the trained model. The Predictor object computes the most likely next value in the sequence based on the current input and the model's learned patterns. The Predictor class acts as a bridge between the trained model and the input sequences, providing a mechanism to generate predictions for future elements in the sequence based on the model's learned patterns.

III. Methodology

This paper aims to implement the application of multi-sequence learning techniques to generate a predicting code based on song or story text that represents the completion engine as used by GPT with prediction accuracy in C#. NET Core. This paper also poses a new method implemented for learning data from a text file. The data will be divided into two parts with 90 percent of it, used for training and 10 percent for testing purposes. Upon initial training with the provided dataset, the model will compute binary cross-entropy loss. Following the completion of the learning process, the model's predictive accuracy will be assessed using separate data from another file or user input.

A. Input Data Preparation

Input data is sourced from two text files: 'trainData.txt' and 'testData.txt'.The content of these files is read and processed to remove unwanted characters (e.g., tabs). Each character is converted to its ASCII value and stored in a list ('inputValues'). The entire dataset is structured so that each 8-character segment overlaps by 4 characters with the adjacent segments before it proceeds to subsequent usage.

B. Setup, Dependencies and Configuration

The program utilizes the NeoCortexApi library, which provides classes and methods for implementing HTM algorithms.It imports necessary name spaces for HTM components such as classifiers, encoders, entities, and networks. The program initializes configuration parameters for the HTM algorithm, such as the number of input bits, number of columns, and various HTM configuration settings.It also configures an encoder to encode scalar values into input representations suitable for HTM processing.

C. Experiment Execution Workflow

The 'Run' method serves as the entry point for executing the sequence learning experiment. It initializes the HTM configuration, encoder, and input values, and then invokes

the ‘RunExperiment‘ method to perform the experiment.The ‘RunExperiment‘ method orchestrates the learning process and consists of several stages:

1) Initialization:

It initializes necessary components such as the HTM connections, classifier, cortical layer, spatial pooler, and temporal memory.

2) Newborn Stage:

Initially, the spatial pooler is trained without temporal memory to stabilize its activity.The program iterates through cycles until the spatial pooler reaches a stable state.

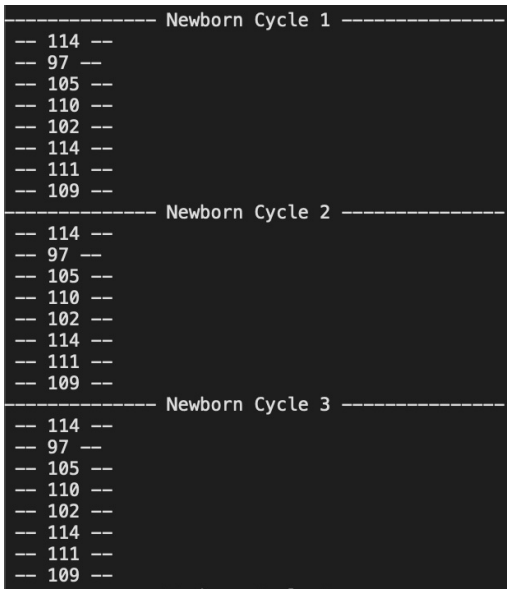

Figure 3. Newborn Cycle

3) Training with SP+TM:

After the newborn stage, the temporal memory is activated, and the system starts training with both spatial pooler and temporal memory. The program iterates through cycles, presenting input values and updating the HTM components accordingly. It tracks matches between predicted and actual sequences, aiming for high accuracy.

4) Sequence Learning Completion:

The program terminates once it achieves a specified level of accuracy for a certain number of cycles, indicating successful sequence learning.It also handles scenarios where the learning process fails to meet the expected accuracy criteria.It tracks matches between predicted and actual sequences, aiming for high accuracy.

D. CalculateBinaryCrossEntropy

The binary cross-entropy loss for each cycle is computed during the training phase by comparing active SDRs with predicted values.

The threshold is established based on the minimum disparity between active SDRs and predicted values. For each comparison, a correctness value is determined, denoted by a binary assignment (1 or 0), indicating whether the prediction aligns within the permissible range defined by the threshold. A correctness value of 1 signifies that the absolute difference falls within the threshold, indicating accuracy, whereas a value of 0 denotes inaccuracy. Following this, the binary cross-entropy values for each prediction are aggregated, with correct predictions assigned a value of 0 and incorrect predictions a value of 1. Subsequently, the total binary cross-entropy is calculated by dividing the sum of these values by the total number of correctness counts.

E. Prediction Accuracy

The testing data, denoted as testData.txt or user input, serves as the basis for predicting the next elements in the sequences. Trained predictors are then applied to this testing data to generate predictions. The accuracy of all these predictions is calculated by tallying the total number of matches, and the prediction with the highest accuracy is favored.

IV. Implementation

This section outlines the execution of the Multi-Sequence Learning Experiment. Our exploration focused on understanding the mechanisms involved in employing multi-sequence learning for numerical sequences and deploying it specifically for text file analysis to assess prediction accuracy. Moreover, the paper presents methods that utilize input from text files for training, calculate binary cross-entropy loss, predict outcomes based on test data or user input (prediction code), and subsequently evaluate prediction accuracy.

A. Training/Learning Phase

The process of learning a sequential arrangement of text characters involves first converting the text into ASCII representation. Subsequently, datasets are initialized based on this representation. These datasets serve as the foundation for training the spatial pooler using HTM setup parameters, a process repeated across multiple iterations. Over time, through this repetitive training, the spatial pooler gradually converges to a stable state, optimizing its performance. This iterative refinement ultimately enhances the model’s ability to process and understand text sequences effectively.

B. Binary Cross Entropy Loss

The methods CalculateBinaryCrossEntropy() and CalculateCorrectness() are used to calculate binary cross entropy loss for each iteration in learning/training phase.[13] This loss function effectively transforms the neural network’s output into a probability distributions. The average of these least binary cross-entropy value is computed by dividing the value of totalBCE, with the total number of cycles.

double aveBCE = totalBCE / cycle;

*accuracyFromBinaryCrossEntropy=(1-aveBCE)*100.0;*

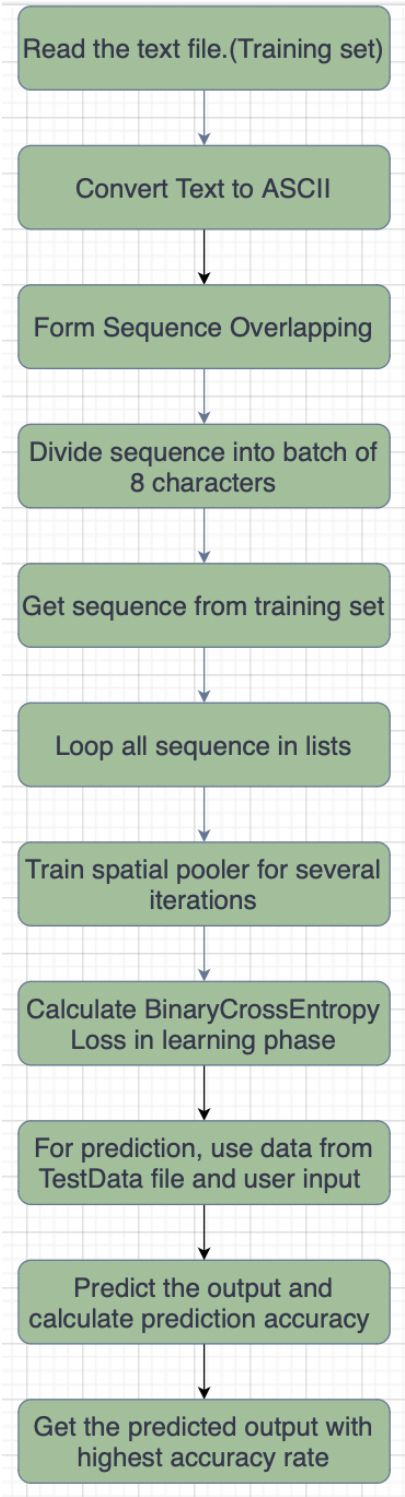

Figure 4. Implementation architecture

Then accuracyFromBinaryCrossEntropy is compared with maxPossibleAccuraccy, where maxPossibleAccuraccy is defined by:

*double maxPossibleAccuracy= (double) ((double)inputValues.Count-1)/(double)inputValues.Count*100;*

```
      // Calculate binary cross entropy
static double
    CalculateBinaryCrossEntropy(List<int>
    actualOutputs, List<int> predictedValues,
    int threshold)
{
 // Determine correctness based on threshold
 var correctness =
     actualOutputs.Zip(predictedValues,
     (actual, pred) => Math.Abs(actual - pred)
     <= threshold ? 1 : 0).ToList();
 // Compute binary cross-entropy
 double bce=
     CalculateBinaryCrossEntropy1(correctness);
 return bce;
}
static double
    CalculateBinaryCrossEntropy1(List<int>
    correctness)
{
    // Compute binary cross-entropy
    double bce = 0;
    foreach (var c in correctness)
       {
        bce += c == 1 ? 0 : 1;
       }
   return bce / correctness.Count;
}
```

C. Prediction and Accuracy calculation

During the inference phase, the HTM model predicts future states of the input data based on the learned patterns and sequences. PredictNextElement() method and Predictor class is used for prediction. Testing data (‘testingData‘) / user input is used to predict the next elements in the sequences.The accuracy is calculated based on the number of successful matches between predicted and total predictions.The output is determined by selecting the string with the highest accuracy from all the predictions made.

*Accuracy = totalMatches/total number of predictions*100*

V. Results

This project outlays text data for multi-sequence learning, and predictions are generated for each batch of input using either training data and user input.
The prediction results are calculated after multiple runs with different datasets. Training the model on multiple dataset is just one way of making a robust system. The accuracy is computed for each cycle, and the final prediction output is determined by selecting the sequence with the highest prediction accuracy. Smaller dataset size has been deliberately used to train and predict accuracy, owing to longer time consumption of the underlying processes. This approach allows us to strike a balance between model performance and computational resources. One of the sample result is captured and shown as a screen-grab image.

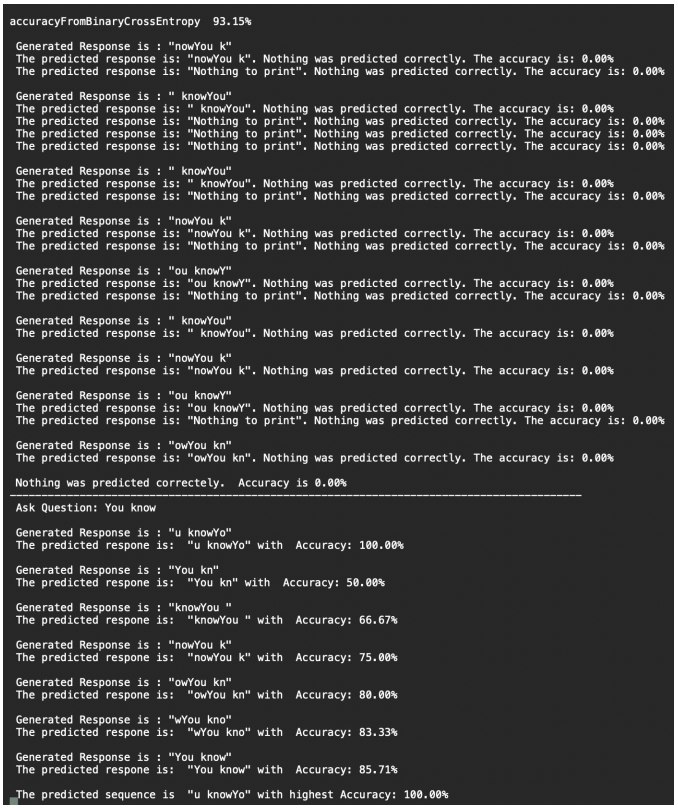

Figure 5. Accuracy Prediction

The dataset below has been employed to obtain the output(Fig 6) for the first batch of TrainData.

TrainData : You know the day destroys the night
TestData: the night is dark
(User Input)Ask Question: You know
Expected Output: You know the
Actual Output: u knowYo

The experiment flow is such that, the data in the TrainData.txt file undergoes processing by removing control characters. And then, the ASCII characters are divided into batches of 8 characters, with each segment overlapping by 4 characters with its adjacent segments. These batches of 8 characters are then passed to the Run() method. The testing data is directly utilized for prediction initially and the output is shown in the first part of the Fig 5 for the above mentioned TestData. Subsequently, user input is obtained for each iteration and passed to the PredictNextElement() method for calculation of prediction accuracy and the output is shown in the second part of Fig 5.
In this project, binary cross-entropy is calculated during the learning phase by comparing the active SDRs with the predicted values for each cycle.

As shown in figure 6, Active SDRs and Predicted values are derived, subsequently determining the minimum difference between the actual(Active SDRs) and predicted values to

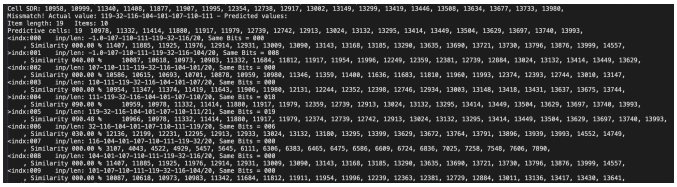

Figure 6. Active SDRs and Predicted Values

establish a threshold, which has been determined to be 15. Subsequently, correctness is calculated for each pair of actual and predicted values. If the difference falls within the threshold, the prediction has been considered correct (assigned as 1); otherwise, it has been deemed incorrect (assigned as 0).

After passing the correctness list to the function CalculateBinaryCrossEntropy(), the following computation was performed:

CalculateBinaryCrossEntropy(correctness):

```
bce = 0;
foreach c in correctness:
     if c == 1:
       bce += 0
     else:
       bce += 1
  return bce / length(correctness)
```

Then, the average binary cross-entropy (aveBCE) was calculated as:

double aveBCE = totalBCE / cycle;

Subsequently, the accuracy from binary cross-entropy (accuracyFromBinaryCrossEntropy) was computed using the formula:
accuracyFromBinaryCrossEntropy=(1-aveBCE)*100.0;
The experimental testing duration, manually conducted over multiple runs, is recorded as 11 minutes and 33 seconds(on an average), using the dataset specified below.
TrainData: You know the day
TestData: Day and night

VI. Unit Testing

This research paper outlines a unit testing framework devised for evaluating Multi Sequence Learning (MSL) models that incorporate language semantics in natural language processing (NLP) tasks. The MS Unit Test framework is designed to assess the robustness and efficacy of MSL models in capturing semantic relationships across multiple sequences of text. It comprises a comprehensive set of test cases, including synthetic and real-world datasets, aimed at evaluating various performance metrics such as accuracy and robustness.

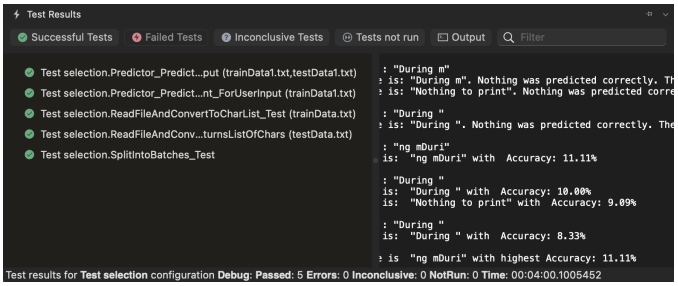

Figure 7. Unit Test Results

VII. Applications

A. Text Generation and Completion

The program can be used to generate or complete text based on input sequences.[1] For example:

- Writing assistants: Generating suggestions for completing sentences or paragraphs. Creative writing tools: Providing inspiration or generating new ideas based on existing text. Chatbots: Generating conversational responses based on user inputs.

B. Language Modeling and Prediction

The program can be applied in language modeling tasks, where the goal is to predict the next word or sequence of words in a sentence.

Applications include: Auto-completion features in text editors or messaging apps. Machine translation systems to predict the next word or phrase in the translated text.

C. Content Recommendation and Personalization

By learning from large datasets of text, the program can provide personalized recommendations for content consumption.

D. Sentiment Analysis and Opinion Mining

The program can analyze text data to determine sentiment and extract opinions or emotions expressed in the text.

E. Text Classification and Topic Modeling

The program can classify text documents into predefined categories or extract topics from unstructured text data.
Overall, the ”Multisequence with language semantic” program can be refined further for analyzing and generating text data, with applications ranging from natural language processing and understanding to content recommendation and personalization. Its flexibility and adaptability make it suitable for a wide range of text-related tasks and applications across various industries.

VIII. Limitations

Along with many benefits that makes the use of MultiSequence learning with language semantic, many limitations also arise in context with the current developments. Some of them can be:

A. Data Dependency

The effectiveness of the program heavily depends on the quality and quantity of the training data. Insufficient or biased training data may lead to inaccurate predictions and limitations in text generation.

B. Overfitting

There's a risk of overfitting to the training data, especially when using complex models with a large number of parameters. Overfitting can lead to poor generalization and limited performance on unseen data.

C. Vocabulary Coverage

The program's performance may be limited by its vocabulary coverage. Words or phrases that are rare or domain-specific may not be adequately represented in the training data, leading to difficulties in prediction and generation.

D. Semantic Understanding

While the program can learn sequences of text, it may lack deep semantic understanding of the language. It may struggle with tasks requiring nuanced comprehension, context understanding, and reasoning.

E. Computational Resources

Training and running the program may require significant computational resources, especially for large-scale datasets and complex models. Limited computational resources can restrict the program's scalability and performance.

F. Evaluation Metrics

Assessing the program's performance and accuracy can be challenging. Evaluation metrics such as binary cross-entropy may not fully capture the quality of generated text or the effectiveness of predictions.

IX. Conclusion

The project utilized the Neocortex API's multi-sequence learning capabilities to analyze sequential text datasets. The Multisequence Learning program, along with its language semantic extension, signifies a notable advancement in applying HTM algorithms to sequence learning and natural language processing tasks. These programs offer solutions for understanding sequential patterns for learning and predicting sequences across domains, its language semantic extension enhances capabilities in natural language processing tasks like text generation and completion. Despite facing challenges such as handling long sequences, and understanding nuanced contextual information, these programs contribute significantly to advancing in reading text data from file, converting it into ASCII values, and allowing the model to run with multiple batches for several iterations.
Binary cross entropy loss is calculated during training phase. Prediction accuracy is calculated by picking out the highest accuracy among all the predictions generated.Further research is needed to address limitations and enhance their effectiveness in real-world scenarios.

References

[1] Alzhrani, K., Alrasheedi, F. S., Kateb, F. A., and Boult, T. E. (2019). Cnn with paragraph to multi-sequence learning for sensitive text detection. In *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, pages 1–6.

[2] Chen, X. and Wang, W. (2012). An overview of hierarchical temporal memory: A new neocortex algorithm. pages 1004–1010.

[3] Cui, Y., Ahmad, S., and Hawkins, J. (2017). The htm spatial pooler—a neocortical algorithm for online sparse distributed coding. *Frontiers in Computational Neuroscience*, 11.

[4] Dauletkhanuly, Y., Krestinskaya, O., and James, A. P. (2020). *HTM Theory*, pages 169–180. Springer International Publishing, Cham.

[5] Dobric, D., Pech, A., Ghita, B., Wennekers, T., et al. (2021). Improved htm spatial pooler with homeostatic plasticity control. SCITEPRESS-Science and Technology Publications.

[6] Hawkins, J. and Ahmad, S. (2016). Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10.

[7] Jiang, L. P. and Rao, R. P. (2024). Dynamic predictive coding: A model of hierarchical sequence learning and prediction in the neocortex. *PLOS Computational Biology*, 20(2):e1011801.

[8] Khan HM, Khan FM, K. A. A. M. A. D. (2021). Anomalous behavior detection framework using htm-based semantic folding technique. comput math methods med.

[9] Main, L. and Thornton, J. (2017). Stable sparse encoding for predictive processing. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8.

[10] Pech, A., Ghita, B., and Wennekers, T. (2020). Scaling the htm spatial pooler. *International Journal of Artificial Intelligence Applications*, 11:83–100.

[11] Pech, A., Ghita, B., and Wennekers, T. (2021). Improved htm spatial pooler with homeostatic plasticity control. pages 98–106.

[12] Putic, M., Varshneya, A., and Stan, M. R. (2017). Hierarchical temporal memory on the automata processor. *IEEE Micro*, 37(1):52–59.

[13] Qu, Z., Mei, J., Liu, L., and Zhou, D.-Y. (2020). Crack detection of concrete pavement with cross-entropy loss function and improved vgg16 network model. *IEEE Access*, PP:1–1.

[14] Ran, Y. and Han, H. (2020). Text classification algorithm based on sparse distributed representation. In *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*, pages 876–880.