Approve Prediction of Multisequence Learning

Muhammad Jasim Suhail
muhammad.suhail @stud.fra-uas.de

Syed Faizan Ul Hassan Zahidi
syed.zahidi@stud.fra-uas.de

***Abstract*— Current artificial networks mostly rely on dense representations, whereas biological networks rely on sparse representations. Hierarchical Temporal Memory is a specific realization of the Thousands Brains Theory that generates motor commands for interacting with the environment and testing predictions. In our research paper, we demonstrate how a sequence is learned using Multsequence Learning algorithms. A subsequence is used to calculating accuracy and how we calculate the accuracy. To perform the experiment, some tasks have been automated such as creating synthetic dataset as per configuration, saving the dataset to file, reading dataset and writing the results to file are performed to support our results.**

Keywords—Hierarchical Temporal Memory, Spatial Pooler, Temporal Memory, encoder, sparse dense representation, AI, ML, HTM .

I. INTRODUCTION

Multi-sequence learning using HTM involves training the algorithm to recognize and predict patterns across multiple input sequences. To implement multi-sequence learning using HTM, we first need to encode the input data into Sparse Distributed Representations (SDRs), which can be done using scalar encoder. Once the data is encoded, the spatial pooler creates sparse representations of the input sequences, which are then fed into the temporal memory component for learning and prediction. Multisequence learning using HTM is a powerful approach for recognizing and predicting patterns across multiple input sequences.

In this project, we have tried to implement new methods along Multisequence Learning algorithm [1]. The new methods are automatically reading the dataset from the given, we also have test data in other file which needs to be read for later testing the subsequence in similar. Multisequence Learning takes the multiple sequences and test subsequence and for learning. After learning is completed, calculation of accuracy of predicted element.

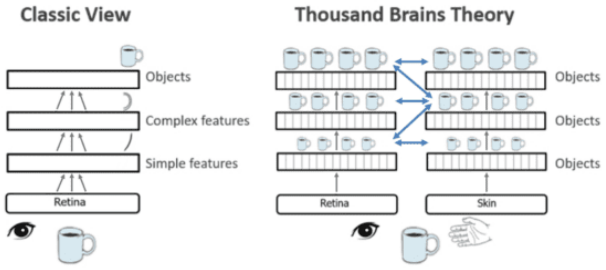

Figure 1: Numeta's machine learning guide to HTM [2]

II. LITERATURE SURVEY

A. Hierarchical Temporal Memory

The objective of HTM is to emulate the hierarchical structure and learning process of the brain. It is comprised of a network of nodes organized hierarchically, where each node corresponds to a group of neurons in the neocortex. These nodes acquire the ability to identify patterns in sensory information and generate predictions based on their prior experiences. Subsequently, the predictions are evaluated against the input data to refine the node's model and enhance its predictive precision.

As per Hawkins, the neocortex learns and makes predictions by forming a hierarchical structure of columns, each containing a set of neurons that recognize patterns in sensory input. These columns communicate with each other in a hierarchical manner, with higher-level columns representing more abstract concepts [3].

B. Sparse Distributed Representation

In the context of HTM, SDRs are used to represent patterns of activity in the network. Each input to the network is transformed into an SDR, which is then processed by the network's hierarchy of nodes to make predictions about future input.

Hawkins and Ahmad proposed that SDRs, which are binary vectors with a small number of active bits (ones) out of a large number of total bits, are a natural way to represent sparse, distributed patterns of activity in the neocortex [4].

C. Encoder

The encoder is designed to take in raw data inputs and convert them into SDRs that represent the relevant features of the input data. The encoding process involves several steps, including dimensionality reduction, noise reduction, and normalization. The encoder also learns to recognize patterns in the input data and adapts to changes in the input distribution over time.

The SP encoder is designed to handle temporal data and uses a sliding window approach to capture temporal patterns in the input data. It first converts the input data into a continuous stream of binary values, which are then fed into the HTM network as a sequence of SDRs [5].

D. Spatial Pooler

The Spatial Pooler is a type of unsupervised learning algorithm that takes in high-dimensional input data and creates a lower-dimensional SDR that represents the relevant features of the input data. It does this by first assigning a random set of weights to each input feature and then computing the overlap between each input and the set of weights. The features with the highest overlap are then selected and included in the SDR [6].

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 [6].

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.

F. Multisequence Learning

Multisequence learning is a HTM based algorithm that involves learning and predicting multiple sequences of patterns simultaneously. Multisequence learning is achieved by using separate Temporal Memory modules to learn and predict each sequence of patterns.

The key idea behind this approach by Ahmad [7] is to use a hierarchical structure of nodes to learn and predict sequences of patterns at different levels of abstraction. At the lowest level, each Temporal Memory module learns and predicts the raw sensory input from a single modality. At higher levels, the nodes learn and predict sequences of patterns that combine information from multiple modalities.

III. METHODOLOGY

Our main goal was to automate couples of process used outside of Multsequence Learning. One of them was to read the input from the file. Read the subsequence from the file. Test the subsequence on the trained model and calculate the accuracy. Along with all these we made the experiment more dynamic to create synthetic dataset. In this method we automate the process for generating multiple sequences required for training purposes.

A. Sequence

Sequence is data model which consists of sequence name and numeric sequence. The object of list of sequence data model is nothing but multiple sequences. The subsequence is subset of sequence and hence used same data model.

```
public class Sequence
{
   public String name { get; set; }
   public int[] data { get; set; }
}
```

Source code 1: Data model of Sequence

```
[
  {
    "name": "S1",
    "data": [ 0, 2, 5, 6, 7, 8, 10, 11, 13 ]
  },
  {
    "name": "S2",
    "data": [ 1, 2, 3, 4, 6, 11, 12, 13, 14 ]
  },
  {
    "name": "S3",
    "data": [ 1, 2, 3, 4, 7, 8, 10, 12, 14 ]
  }
]
```

Dataset 1: Sample dataset

```
[
  {
    "name": "T1",
    "data": [ 1, 2, 4 ]
  },
  {
    "name": "T2",
    "data": [ 2, 3, 4 ]
  },
  {
    "name": "T3",
    "data": [ 4, 5, 7 ]
  },
  {
    "name": "T4",
    "data": [ 5, 8, 9 ]
  }
]
```

Dataset 2: Sample subsequence

B. FetchHTMConfig method

Here we save the HTMConfig which is used for Hierarchical Temporal Memory to Connections.

```
/// <summary>
/// HTM Config for creating Connections
/// </summary>
/// <param name="inputBits">input bits</param>
/// <param name="numColumns">number of columns</param>
/// <returns>Object of HTMConfig</returns>
public static HtmConfig FetchHTMConfig(int inputBits, int numColumns)
{
   HtmConfig cfg = new HtmConfig(new int[] { inputBits }, new int[] { numColumns })
   {
      Random = new ThreadSafeRandom(42),

      CellsPerColumn = 25,
      GlobalInhibition = true,
      LocalAreaDensity = -1,
      NumActiveColumnsPerInhArea = 0.02 * numColumns,
      PotentialRadius = (int)(0.15 * inputBits),
      MaxBoost = 10.0,
      DutyCyclePeriod = 25,
      MinPctOverlapDutyCycles = 0.75,
      MaxSynapsesPerSegment = (int)(0.02 * numColumns),
      ActivationThreshold = 15,
      ConnectedPermanence = 0.5,e.
      PermanenceDecrement = 0.25,
      PermanenceIncrement = 0.15,
      PredictedSegmentDecrement = 0.1,
   };

   return cfg;
}
```

Source code 2: FetchHTMConfig Method

Through this provision, multiple configs can be created and different models can be trained. This gives some space to scale up the experiment.

C. GetEncoder Method

We have used ScalarEncoder [1] since we are encoding all numeric value only.

```
/// <summary>
/// Get the encoder with settings
/// </summary>
/// <param name="inputBits">input bits</param>
/// <returns>Object of EncoderBase</returns>
public static EncoderBase GetEncoder(int inputBits)
{
      double max = 20;

      Dictionary<string, object> settings = new Dictionary<string, object>()
      {
      { "W", 15},
      { "N", inputBits},
      { "Radius", -1.0},
      { "MinVal", 0.0},
      { "Periodic", false},
      { "Name", "scalar"},
      { "ClipInput", false},
      { "MaxVal", max}
      };

      EncoderBase encoder = new ScalarEncoder(settings);

      return encoder;
}
```

Source code 3: Implementation of Scalar Encoder method

D. ReadDataset method

Reads the JSON file wehen passed as full path and retuns the object of list of Sequence data model.

```
/// <summary>
/// Reads dataset from the file
/// </summary>
/// <param name="path">full path of the file</param>
/// <returns>Object of list of Sequence</returns>
public static List<Sequence> ReadDataset(string path)
{
      Console.WriteLine("Reading Sequence...");
      String lines = File.ReadAllText(path);

      List<Sequence> sequence =
System.Text.Json.JsonSerializer.Deserialize<List<Sequ
ence>>(lines);

      return sequence;
}
```

Source code 4: Method to read dataset file

E. CreateDataset method

We made an enhancement to create dataset automatically, so we do not have to manually spend time. Here we create dataset with paremeters such as numberOfSequence to be created, size of a sequence, startVal possibly start range, and endVal possibly start range of sequence.

```
/// <summary>
/// Creates list of Sequence as per configuration
/// </summary>
/// <returns>Object of list of Sequence</returns>
public static List<Sequence> CreateDataset()
{
      int numberOfSequence = 3;
      int size = 12;
      int startVal = 0;
      int endVal = 15;
      Console.WriteLine("Creating Sequence...");
      List<Sequence> sequence =
HelperMethods.CreateSequences(numberOfSequence,
size, startVal, endVal);

      return sequence;
}
```

Source code 5: Create list of sequence as per configuration.

Note that endVal should be less than equal to MaxVal of ScalarEncoder used above.

F. SaveDataset method

Saves the dataset in dataset director of the BasePath of the application where it is running.

```
/// <summary>
/// Saves the dataset in 'dataset' folder in BasePath of
application
/// </summary>
/// <param name="sequences">Object of list of
Sequence</param>
/// <returns>Full path of the dataset</returns>
public static string SaveDataset(List<Sequence>
sequences)
{
      string BasePath =
AppDomain.CurrentDomain.BaseDirectory;
      string reportFolder = Path.Combine(BasePath,
"dataset");
      if (!Directory.Exists(reportFolder))
      Directory.CreateDirectory(reportFolder);
      string reportPath = Path.Combine(reportFolder,
$"dataset_{DateTime.Now.Ticks}.json");

      Console.WriteLine("Saving dataset...");

      if (!File.Exists(reportPath))
      {
      using (StreamWriter sw =
File.CreateText(reportPath))
      {

sw.WriteLine(JsonConvert.SerializeObject(sequences));
      }
      }

      return reportPath;
}
```

Source code 6: Saves the dataset in JSON file.

G. Calculate accuracy in PredictNextElement method

```
int matchCount = 0;
int predictions = 0;
double accuracy = 0.0;

foreach (var item in list)
{
   Predict();
   //compare current element with prediction of previous
element
   if(item == Int32.Parse(prediction.Last()))
   {
      matchCount++;
   }
   predictions++;
   accuracy = (double)matchCount / predictions * 100;
}
```

Source code 7: Calculate accuracy

IV. RESULTS

We have run the experiment max possible number of times with different dataset. We have tried to keep the size of dataset small and number of sequences also small due to large time in execution.

```
------------------------------
Sequence: S1 -> 0-2-5-6-7-8-10-11-13
Sequence: S2 -> 1-2-3-4-6-11-12-13-14
Sequence: S3 -> 1-2-3-4-7-8-10-12-14
------------------------------
Using test sequence: T4 -> 3-4-7-8-10
    Input: 3, Predicted Sequence: S2, Predicted next element: 4
    Input: 4, Predicted Sequence: S3, Predicted next element: 7
    Input: 7, Predicted Sequence: S3, Predicted next element: 8
    Input: 8, Predicted Sequence: S3, Predicted next element: 10
    Accuracy: 100%
------------------------------
Using test sequence: T4 -> 3-4-7-8-10
    Input: 3, Predicted Sequence: S2, Predicted next element: 4
    Input: 4, Predicted Sequence: S3, Predicted next element: 7
    Input: 7, Predicted Sequence: S3, Predicted next element: 8
    Input: 8, Predicted Sequence: S3, Predicted next element: 10
    Accuracy: 100%
------------------------------
Using test sequence: T4 -> 3-4-7-8-10
    Input: 3, Predicted Sequence: S2, Predicted next element: 4
    Input: 4, Predicted Sequence: S3, Predicted next element: 7
    Input: 7, Predicted Sequence: S3, Predicted next element: 8
    Input: 8, Predicted Sequence: S3, Predicted next element: 10
    Accuracy: 100%
------------------------------
Using test sequence: T4 -> 3-4-7-8-10
    Input: 3, Predicted Sequence: S2, Predicted next element: 4
    Input: 4, Predicted Sequence: S3, Predicted next element: 7
```

Figure 2: Prediction and calculation of accuracy on subsequence

V. DISCUSSION

We can improve more by creating test data which is subsequence out of synthetic dataset which is created. This will assure that the test dataset is 100% a subsequence and can match the accuracy.

Run the experiment on cloud to run with more and large number of inputs so that the subsequences are also more.

VI. REFERENCES

[1] "NeoCortexApi : https://github.com/ddobric/neocortexapi".

[2] “Numenta https://www.numenta.com”.

[3] Jeff Hawkins, "On Intelligence: How a New Understanding of the Brain will Lead to the Creation of Truly Intelligent Machines", 2004

[4] Jeff Hawkins and Subutai Ahmad, "Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex", 2016

[5] Subutai Ahmad, "Hierarchical Temporal Memory" - Scholarpedia, 2012

[6] Jeff Hawkins and Dileep George, "Hierarchical Temporal Memory", 2009

[7] Subutai Ahmad, "Real-Time Multimodal Integration in a Hierarchical Temporal Memory Network", 2015