Information Technology Course
Module Software Engineering
by Damir Dobric

Approve Prediction of Multisequence Learning

Gaurav Kale
gaurav.kale@stud.fra-uas.de

***Abstract*— Contemporary artificial networks predominantly utilize dense representations, contrasting with biological networks that favor sparse representations. Hierarchical Temporal Memory (HTM) represents a specific implementation of the Thousand Brains Theory, which generates motor commands for environmental interaction and hypothesis testing. In our research paper, I illustrate the learning process of sequences through Multisequence Learning algorithms. I use subsequences to compute accuracy and detail our methodology for accuracy calculation. To facilitate our experimentation, I have automated several tasks, including generating synthetic datasets based on specified configurations, saving datasets to files, and reading datasets, as well as writing results to files, all of which contribute to supporting our findings.**

INTRODUCTION

Employing Hierarchical Temporal Memory (HTM) for multi-sequence learning entails training the algorithm to discern and forecast patterns across numerous input sequences. To initiate multi-sequence learning with HTM, the initial step involves encoding the input data into Sparse Distributed Representations (SDRs), accomplished through a scalar encoder. Subsequently, the spatial pooler generates sparse representations of the input sequences, which are subsequently processed by the temporal memory component for learning and predictive analysis. Leveraging HTM for multi-sequence learning offers a robust method for identifying and forecasting patterns across diverse input sequences.

In this project, I endeavoured to integrate novel methods within the Multisequence Learning algorithm [1]. These methods automate the dataset reading process from a specified location. Additionally, I possess separate test data in another file, essential for subsequent evaluation of subsequence's. Multisequence Learning involves processing multiple sequences alongside test subsequence's for the learning phase. Upon completion of the learning process, accuracy calculation of predicted elements follows.

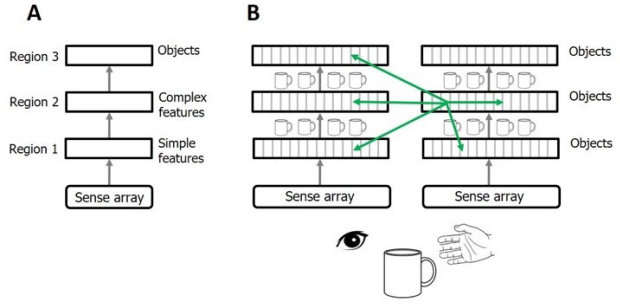

Figure 1: Numeta's machine learning guide[2]

I. LITERATURE SURVEY

A. Hierarchical Temporal Memory

The primary goal of HTM is to replicate the hierarchical organization and learning mechanisms observed in the brain. It consists of a network of nodes arranged hierarchically, where each node corresponds to a cluster of neurons within the neocortex. These nodes develop the capability to detect patterns in sensory data and make predictions based on past experiences. These predictions are then compared to incoming data to refine the node's model and improve its predictive accuracy.

According to Hawkins, the neocortex learns and predicts by establishing a hierarchical arrangement of columns. Each column comprises a group of neurons specialized in recognizing patterns within sensory input. These columns interact with one another hierarchically, with higher-level columns representing more abstract concept *Sparse Distributed Representation*

B. Sparse Distributed Representation

Within HTM, Sparse Distributed Representations (SDRs) serve as the means to depict activity patterns across the network. Every input fed into the network undergoes transformation into an SDR, subsequently analyzed by the network's hierarchical nodes to forecast forthcoming input. Hawkins and Ahmad asserted that SDRs, binary vectors characterized by few active bits amid a vast array of total bits, provide an inherent mechanism for delineating sparse, distributed activity patterns within the neocortex.

C. Spatial Pooler

The Spatial Pooler, categorized as an unsupervised learning algorithm, processes high-dimensional input data to generate a lower-dimensional Sparse Distributed Representation (SDR) that encapsulates the pertinent features of the input. Initially, it assigns random weights to each input feature and subsequently calculates the overlap between every input and this set of weights. The features exhibiting the greatest overlap are then chosen and incorporated into the SDR.

D. Temporal Memory

The operation of Temporal Memory revolves around maintaining a group of active cells that encapsulate the ongoing context of the input data. Upon receiving new input

patterns, these active cells undergo adjustments depending on the resemblance between the input and the existing context. Furthermore, the algorithm manages a network of connections among cells, representing the sequence of encountered patterns[3].

Leveraging these connections, Temporal Memory can anticipate the subsequent probable input pattern based on the current context. In the event of accurate prediction, the algorithm strengthens the connections among the cells that were active during the predicted sequence. Conversely, if the prediction is inaccurate, the algorithm modifies the connections to diminish the likelihood of such a sequence occurring in the future.

E. Multisequence Learning

Multisequence learning, an HTM-based algorithm, entails the simultaneous learning and prediction of multiple sequences of patterns. This is accomplished by employing distinct Temporal Memory modules to handle the learning and prediction tasks for each sequence of patterns.

The fundamental concept, as proposed by Ahmad, involves employing a hierarchical arrangement of nodes to learn and forecast sequences of patterns at various levels of abstraction. At the lowest tier, each Temporal Memory module learns and predicts raw sensory input from a single modality. As we ascend through higher tiers, nodes are tasked with learning and predicting sequences of patterns that amalgamate information from multiple modalities[4][5].

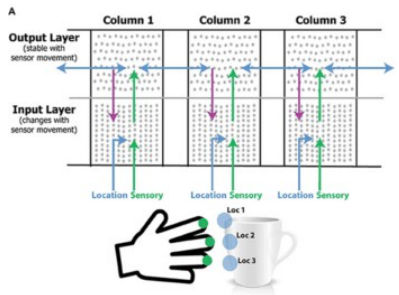

Figure 2: Columns in Neocortex Enable Learning the Structure[2].

II. METHODOLOGY

A. My primary objective was to automate several processes utilized alongside Multisequence Learning. Among these tasks were reading input from files, extracting subsequences from files, testing these subsequences on the trained model, and computing accuracy. Additionally, I enhanced the experiment's dynamism by enabling the creation of synthetic datasets. Through this method, I automated the generation of multiple sequences necessary for training.

B. Sequence

A sequence is a data model comprising a sequence name and a numeric sequence. A list of sequence data models represents multiple sequences. A subsequence, being a subset of a sequence, utilizes the 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

C. FetchHTMConfig method

Here I saved the HTMConfig which is used for Hierarchical Temporal Memory to Connections.

```
/// <summary>
/// HTM Config for creating Connections
/// <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.

D. GetEncoder Method

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

```
/// <summary>
/// Get the encoder with settings
/// <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

E. ReadDataset method

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

```
/// <summary>
/// Reads dataset from the file
/// <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

F. CreateDataset method

We've implemented an improvement to automate dataset creation, eliminating the need for manual effort. This enhancement allows for the creation of datasets based on specified parameters such as the number of sequences to be generated, sequence size, and optionally, startVal and endVal to define the range of values for the sequence.

```
/// <summary>
/// Creates list of Sequence as per configuration
/// <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.

G. 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
/// <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.

H. 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

III. RESULTS

As, I 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.

```
Input: 6
Nothing predicted :(
Input: 7
Predicted Sequence: S23 - Predicted next element: 9
------------------------------
Accuracy for T1 sequence: 33.33333333333333%
------------------------------
Using test sequence: T2
------------------------------
Input: 6
Predicted Sequence: S30 - Predicted next element: 8
Input: 11
Predicted Sequence: S29 - Predicted next element: 12
Input: 12
Predicted Sequence: S29 - Predicted next element: 14
------------------------------
Accuracy for T2 sequence: 33.33333333333333%
------------------------------
Using test sequence: T3
------------------------------
Input: 1
Predicted Sequence: S27 - Predicted next element: 3
Input: 2
Predicted Sequence: S30 - Predicted next element: 3
Input: 3
Predicted Sequence: S30 - Predicted next element: 4
------------------------------
Accuracy for T3 sequence: 66.66666666666666%
------------------------------
Using test sequence: T4
------------------------------
Input: 3
Predicted Sequence: S24 - Predicted next element: 4
Input: 4
Predicted Sequence: S30 - Predicted next element: 6
Input: 7
Predicted Sequence: S23 - Predicted next element: 9
Input: 8
Predicted Sequence: S30 - Predicted next element: 9
------------------------------
Accuracy for T4 sequence: 25%
```

Figure 1

```
------------------------------
Using test sequence: T4
------------------------------
Input: 0
Predicted Sequence: S30 - Predicted next element: 1
Input: 0
Predicted Sequence: S30 - Predicted next element: 1
Input: 0
Predicted Sequence: S30 - Predicted next element: 1
Input: 0
Predicted Sequence: S30 - Predicted next element: 1
------------------------------
Accuracy for T4 sequence: 0%
Done...

Saving session...
...copying shared history...
...saving history...truncating history files...
...completed.

[Process completed]
```

IV. FIGURE 1, 2: PREDICTION AND CALCULATION OF ACCURACY ON SUBSEQUENCE

V. DISCUSSION AND CONCLUSION

Further improvement can be achieved by generating test data as subsequences extracted from the synthetic dataset. This approach ensures that the test dataset is exclusively composed of subsequences, guaranteeing accurate matching during accuracy assessment.

VI. REFERENCES

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

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

[3].https://www.numenta.com/technology/research/thousand-brains-theory/

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

[5]. Jeff Hawkins and Subutai Ahmad, "Why Neurons Have Thousands of Synapses, A Theory of Sequence Memory in Neocortex", 201