*ML22/23-12: Implement Anomaly Detection Sample*

|  |  |
| --- | --- |
| Md Sohel Rana  sohel.rana@stud.fra-uas.de | Md Ashiqur Rahman  ashiqur.rahman@stud.fra-uas.de |

*Abstract*—HTM (Hierarchical Temporal Memory) is an impactful machine learning algorithm approach that is biologically inspired in both aspects, structurally and functionally, by the neocortex of a human brain that processes time series data in a distributed manner using a hierarchical network of nodes. HTM works in a decentralized manner with the help of a tiered arrangement that allows each node and column to learn and recognize patterns in input data. This feature enables actions such as processing information, recognizing and identifying patterns, and making future predictions based on previous learning. This is a potential approach which can be used for anomaly detection and prediction in numerous sectors such as healthcare, finance, geological disasters, cyber-intrusion detection, military surveillance, system fault detection. This paper presents an anomaly detection sample using an HTM model trained on multiple simple numeric integer sequences. This model learns patterns from the input data and identifies anomalies by comparing real data with predicted data from learning within a set tolerance threshold. The paper also provides a detailed explanation of anomaly detection techniques with algorithm implementation.

Keywords—HTM, anomaly detection, machine learning, multi-sequence learning, NeoCortex API.

# Introduction

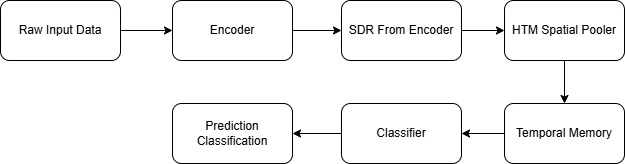
Hierarchical Temporal Memory (HTM), a biologically constrained machine intelligence technique, was created by Numenta. It was first published in 2004 by Sandra Blakeslee and Jeff Hawkins, a brain scientist and the founder of the Redwood Neuroscience Research Institute [1]. This machine learning algorithm works based on the theory of how the biological neocortex works, and this approach basically depends on principles of the Thousand Brains Theory. The fundamental of this approach is responsible for higher order processes like language, conscious movement and thought, and sensory perception [2]. HTM design and operation are modeled after the neocortex, a sizable, intricate region of the human brain. HTM aims to replicate the same fundamental neocortical processes by recognizing complex temporal patterns and correlations in data and making future predictions from them [3].

Hierarchy in HTM refers to the layered structure of a neural network, which consists of multiple layers of neurons. Each layer performs a specific type of computation, and information is passed on from lower layers to higher layers for further processing. The lower layers receive input from the environment, such as sensory data, and encode the input into a distributed representation in these layers. It is especially well-suited for sequence learning modeling, similar to RNN methods, like Long Short-Term Memory (LSTM) and Gated Recurrent unit (GRU) [3].

HTM can be viewed as a specific kind of hierarchical Bayesian model. It also uses spatial-temporal theory to learn the structure and invariance of the space of problems [4]. By following its characteristics HTM has also been applied to anomaly detection in recent years. An anomaly is something that deviates from the typical or expected state. Anomalies are sometimes referred to as outliers, discordant observations, exceptions, aberrations, surprises, etc. Finding anomalous patterns in data is known as anomaly detection. A large portion of the data in the world is time-series, streaming data, and in crucial situations, anomalies can provide important information. Examples of this can be found in a variety of industries, including energy, IT, security, finance, and medicine. It's challenging to find anomalies in streaming data, detectors must process data in real-time rather than in batches and learn while making predictions. The effectiveness of real-time anomaly detectors cannot be sufficiently tested or scored using any benchmarks [5]. The ideal detector would function with real-world time-series data across several domains, identify all abnormalities as quickly as feasible, avoid false alarms, and automatically adjust to changing statistics [4]. Because the Hierarchical Temporal Memory Cortical Learning Algorithm (HTM CLA) has most of the properties, its use in anomaly detection is becoming more and more popular.

In HTM CLA several essential elements are included to handle input data. The raw input data is first encoded and transformed into a sparse distributed representation(SDR) using an encoder. This SDR, which includes binary information with few active bits, is made more robust to noise by passing it via a spatial pooler. The Temporal Memory component, which is responsible for recognizing and detecting patterns in the data, then processes the output. The Classifier component uses these learned patterns to classify input data and predict new patterns. Additionally, over time, the system will continuously learn new patterns thanks to the Homeostatic Plasticity Controller [3].

Figure 1 shows how input data is processed in an HTM system.



**Figure 1: How HTM System works**

The temporal memory layer of the HTM network processes the encoded input data following spatial pooling. Capturing and memorizing temporal patterns and sequences in the input data is the responsibility of temporal memory. By creating predictive links between active cells in various time steps, it keeps a predictive model of the input stream [3].

Based on the present input and past context, the HTM network can predict and infer future states after learning temporal patterns in the input data. To predict the next probable input in the sequence, the network uses the input data to activate predictive cells during inference [3].

HTM systems use feedback mechanisms to continuously learn and adjust to shifting input patterns. The network strengthens its ties and gains knowledge from accurate predictions when the expected and real inputs match. On the other hand, the network adjusts its connections to enhance subsequent forecasts in the event of a prediction error [3].

HTM systems can detect anomalies or departures from expected input patterns in addition to inference and prediction. When the input data deviates substantially from the taught prediction model, it is considered an anomaly and may indicate uncommon or unexpected events [3].

# Methodology

To detect Anomalies in our project we used the Neocortex API [1], which was developed within the .NET framework, we also used Hierarchical Temporal Memory (HTM) for its unique functionality [3]. For training and testing our project, we are going to use artificially generated data, which contains numerous samples of simple integer sequences in the form of (1,2,3,..). These sequences will be placed in a few commas separated value (CSV) files. There will be two folders inside our main project folder, training and predicting. These folders will contain a few of these CSV files. The predicting folder contains data like training, but with added anomalies randomly added inside it. We are going to read data from both the folders and train our HTM model using it. After that we are going to take a part of numerical sequence, trim it in the beginning, from all the numeric sequences of the predicting data and use it to predict anomalies in our data which we have placed earlier, and this will be automatically done, without user interaction.

We are using artificially generated network traffic load data (in percentage, rounded to the nearest integer) from a sample web server. The values are taken over time form numerical sequence .

For our test we will consider values between [45,55] as normal , and anything outside this range as anomalies . The predicting folder contains data with random anomalies placed at different positions, with values between [0,100]. The combined data from both the training folder and predicting folder is shown in Figure 3.

We will use the **MultiSequenceLearning** class from the NeoCortex API as the foundation of our project. It will help us train the HTM model and make predictions. The class works like this:

a. First, HTM configuration is set, and memory connections are initialized. Then, the HTM Classifier, Cortex Layer, and Homeostatic Plasticity Controller are set up.

b. Next, the Spatial Pooler and Temporal Memory are initialized.

c. The spatial pooler memory is added to the cortex layer and trained for the maximum number of cycles.

d. Then, temporal memory is added to the cortex layer to learn all the input sequences.

e. Finally, the trained cortex layer and HTM classifier are returned.

We need to pass Encoder and HTM Configuration settings to the relevant components in this class. We will use the classifier object from the trained HTM model to predict values, which will then be used for anomaly detection.

We will train and test data with integer values ranging from 0 to 100, without periodicity. The configuration settings are shown in Listing 1. We will use 21 active bits for representation. There are 101 values representing integers between 0 and 100. The total input bits are calculated as:

n = buckets + w – 1 = 101 + 21 - 1 = 121.

int inputBits = 121;

int numColumns = 1210;

------------------------------------------------------------------

double max = 100;

Dictionary<string, object> settings = new Dictionary<string, object>()

{

{ "W", 21},

{ "N", inputBits},

{ "Radius", -1.0},

{ "MinVal", 0.0},

{ "Periodic", false},

{ "Name", "integer"},

{ "ClipInput", false},

{ "MaxVal", max}

};

Listing 1: Encoder settings for our project

The minimum and maximum values are set at 0 and 100, as all expected values fall within this range. If the input data has a different range, these values should be adjusted accordingly. We have kept the default HTM configuration unchanged.

Our project follows these steps:

a. The ReadFolder method from the CSVFolderReader class reads all files in a folder. Alternatively, the ReadFile method from CSVFileReader reads a single file. Both store the data as a list of numeric sequences for later use. These classes include exception handling to manage non-numeric data. The TrimSequences method is used in our unsupervised approach. It randomly removes 1 to 4 elements from the start of a numeric sequence and returns the trimmed version. Both methods are shown in Listing 2.

public List<List<double>> ReadFolder()

{

-------

return folderSequences;

}

public static List<List<double>>

TrimSequences(List<List<double>> sequences)

{

-----

return trimmedSequences;

}

Listing 2: Important methods in CSVFolderReader class

b. Next, the BuildHTMInput method from the CSVToHTMInput class converts the read sequences into a format suitable for HTM training. This is shown in Listing 3.

public Dictionary<string, List<double>> BuildHTMInput(List<List<double>> sequences)

{

Dictionary<string, List<double>> dictionary = new Dictionary<string, List<double>>();

for (int i = 0; i < sequences.Count; i++)

{

// Unique key created and added to dictionary for HTM Input

string key = "S" + (i + 1);

List<double> value = sequences[i];

dictionary.Add(key, value);

}

return dictionary;

}

Listing 3: BuildHTMInput method

c. Then, the RunHTMModelLearning method from the HTMModeltraining class trains the model using the MultiSequenceLearning class, as shown in Listing 4. It combines numerical sequences from both the training and predicting folders to train the HTM model. This class returns the trained model object, predictor, which will later be used for prediction and anomaly detection.

MultiSequenceLearning learning = new MultiSequenceLearning();

predictor = learning.Run(htmInput);

Listing 4: Code demonstrating how data is passed to HTM model using instance of class multisequence learning

d. The HTMAnomalyTesting class is used to detect anomalies. It follows these steps:

* The paths of the training and predicting folders are passed to the class constructor.
* The Run method handles the entire process of running the anomaly detection system.
* First, the HTMModelTraining class is used to train the model by passing the folder paths through the constructor.
* Next, the CSVFolderReader class reads the test data from the predicting folder. Before prediction, the TrimSequences method trims 1 to 4 elements randomly from the start of each sequence, as shown in Listing 5. This creates subsequences for testing. The test data contains randomly placed anomalies, which will be detected using the trained model.

# Results

This Part of the text describes results of your works. There can only be mentioned references, MUST point back to Methods and Intro chapter. No more external references.

Code examples must be provided to demonstrate how to use the algorithm/module. Provide a reference to more unit tests, which show the same in more detail. Also provide all diagrams with comments and reference to unit tests, which generate diagrams.

# Discussion

The conclusion of your work should be precise and concise. How was the project, what was done, what was the result. There can be discussion on further work and direction.

##### References

|  |  |
| --- | --- |
| [1] | J. H. a. S. Blakeslee, "On Intelligence," in *Henry Holt*, New York, 2004. |
| [2] | J. S. a. S. Latré, "Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons," in *Neurocomputing*, 2020. |
| [3] | V. Lomonaco, "A machine learning guide to HTM (Hierarchical Temporal Memory)," 2019. [Online]. Available: https://www.numenta.com/blog/2019/10/24/machine-learning-guide-to-htm. |
| [4] | a. H. H. J. A. Starzyk, "Spatio–Temporal Memories for Machine Learning: A Long-Term Memory Organization," in *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 2009. |
| [5] | A. &. A. S. Lavin, "Evaluating real-time anomaly detection algorithms - the Numenta Anomaly Benchmark," in *In arXiv [cs.AI]*, 2015. |