**Multiple Sequences Learning**

The existing multisequence learning application is working with learning/understanding of sequences and the matching sequences, but the application completely works with the hardcoded input sequences.

Team\_MSL will analyse and work on taking the input sequences from any config file can be in any format (e.g. .txt, .csv, .xml, .json, .excel).

Team\_MSL will also work on increasing the efficiency of the code by comparing the different approaches for file operations, along with team will also work on sub sequences matching and prediction.

**Names of Team\_MSL members:**

• Poonam Dashrath Paraskar

• Pratik Desai

• Ankita Talande

**Strategy Team\_MSL is following for code changes and merge is as following:**

1. All the team members will commit their changes in their own feature branch and raise PR for merging those changes into Team\_MSL branch.
2. After reviewing the PR changes team members will merge those changes into Team\_MSL branch.
3. All the team members must take pull of latest Team\_MSL branch before committing any changes and before raising PR.
4. Team members will have to resolve merge conflicts (if any) before raising PR by their own.
5. Once we are done with the development and the Team\_MSL branch is up to date with all the code changes then we will raise PR from Team\_MSL branch to master branch.

**Highlighted Points -**

1. The Hierarchical Temporal Memory (HTM)
2. Spatial Pooler
3. To optimize the sequence learning behaviour of spatial pooler and temporal memory layer in dependence on HTM Sparsity
4. The sequence learning behaviour of spatial pooler and temporal memory layer in dependence on learning parameter-Cells per Column
5. Improve HTM Spatial Pooler with Homeostatic Plasticity Control

**List of tasks done by Team\_MSL per sprint:**

**Sprint 1:**

1. Team\_MSL - forked the Neo-cortex repository.
2. Created new repository out of existing neo-cortex and named as neocortexapi\_Team\_MSL
3. All team members have created their own branches and one common branch for final code changes merge.

**Sprint 2:**

1. All team members analysed the existing multisequence learning project and tried to run it on their local visual studio code.
2. Resolved the build failures and able to run the program and view the output locally.
3. Created a Readme.md file and added basic description about what are all the tasks each team member will perform.

**Sprint 3:**

1. Team started actual development in this sprint and worked on taking the input sequences from local text file.
2. Created new method in Program.cs for eliminating the hard coded inputs and getting from local text file.
3. Able to debug the flow with new input sequences.
4. Team also worked on logic for splitting different sequences.

**Sprint 4:**

1. Created new method – GetInputFromTextFile
2. Implemented new logic for input sequence splitting and testing

**Sprint 5:**

1. Splitted input sequences using regulare expression.
2. Written the logic for subsequence inputs.
3. Created new method – GetSubSequenceInputFromTextFile

**Hierarchical Temporal Memory**

Encoder -

Encoder is chosen according to the type of the inputs. There are some encoders available for popular input type:

- Scalar Encoder

- Datetime Encoder

- Boolean Encoder

- Category Encoder

- Geo-Spatial Encoder

In this project we are using - Scalar Encoder

Scalar Encoder is one of the encoding techniques and is a part of Hierarchical Temporal Memory (HTM). HTM is a machine intelligence technology which is trying to imitate the process and architecture of neocortex. The main purpose for scalar encoder is to encode numeric or floating-point value into an array of bits, where the output has 0’s with an adjacent block of 1’s. The location of the block of 1’s varies continuously depending on the input value.

The scalar representation of value (e.g. for categories, this is the internal index used by the encoder). This number is consistent with what is returned by **[getScalars()](https://nupic.docs.numenta.org/1.0.3/api/algorithms/encoders.html" \l "nupic.encoders.base.Encoder.getScalars" \o "nupic.encoders.base.Encoder.getScalars)**. This value is always an int or float, and can be used for numeric comparisons.

HTM consists of 2 different components: Spatial Pooler and Temporal Memory.

1. Spatial Pooler -

Encoder produces output to be fed into Spatial Pooler algorithm. Type of Spatial Pooler (SP) that is used in this example is the multithreaded version that utilize multicore of the machine to run the spatial pooler algorithm.

SpatialPoolerMT spatialPooler = new SpatialPoolerMT(hpa);

patialPooler.Init(memory, UnitTestHelpers.GetMemory());

1. Temporal Memory -

The output of Spatial Pooler (SDR) is used as the input of Temporal Memory.Temporal memory algorithm will then learn the temporal pattern from spatial pattern.

TemporalMemory temporalMemory = new TemporalMemory();

temporalMemory.Init(mem);