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

Implementation of HTM FeedForward network

Md Mukit Khan
md.mukitkhan@stud.fra-uas.de

S M Mehedi Hasan
s.hasan@stud.fra-uas.de

Abstract— HTM Feedforward network is a multilayer based artificial neural orchestrate which is a biologically propelled show of a single cortical column of the NeoCortex, is a set of six layers of the portion of mammalian cerebral cortex wherever the higher cognitive functioning is acknowledged to originate from. Previous findings in neurosciences show that there is the presence of two sets of Feedforward network in the single cortical column of the human brain among them layer L4-L2 Feedforward network plays the active role to learn new things from the environment. Within the L4-L2 Feedforward network arrange, the lower layer L4 takes the sensory data directly as input from the environment and passes the processed data to layer L2 to perform cognitive predicting & learning functions in the brain. In this paper, the idea to implement the layer L4-L2 based HTM Feed Forward network is demonstrated utilizing the most recent adaptation of NeocortexApi package, which is an open-source solution of the Hierarchical Temporal Memory Cortical Learning Algorithm. Besides, it is also examined how the implemented L4-L2 Feedforward network arrangement behaves at upper layer L2 in case of sequence learning and predicting using HTM Classifier. Aside from that, NuMenta's discoveries and guidelines are investigated as well. The results show that the proposed L4-L2 based HTM Feedforward network with NeocortexApi can learn and predict sequential data patterns with precision in the upper layer region.

Keywords— HTM CLA, Spatial Pooler(SP), Temporal Memory(TM), Sparse Distributed Representation(SDR), Cortical column, Cortical Circuit, FeedForward network, Sequence Learning, Active Cell, Predictive Cell, HTM Classifier

I. INTRODUCTION

HTM is a human brain-inspired unsupervised learning neural network model based on computational principles of single cortical columns of the NeoCortex. [**1**][**4**] HTM gives the most excellent prediction accuracy and exactness. Furthermore, HTM illustrates numerous other features that are alluring for real-world sequence learning, such as quick adjustment to changes within the information stream, robustness to sensor noise and fault tolerance. These highlights make HTM a perfect candidate for sequence learning compare to other neural models [**2**]

In the human brain, a single cortical column of the NeoCortex has a group of mini-columns, Figure 1. And each of these mini-column is divided into six layers. So, conventionally column portions are renamed based on their position on the cortical layer as L1, L2, L3, L4, L5, and L6 column portions. Where L1 column portions are in the most upper layer region and L6 column portions is in the lower layer region of a single cortical column. However, a mini-column has millions of neuron cells, out of them only 0.02% are activated during learning any sensory inputs from the environment. The idea of the HTM Feedforward network originated from the working principle of the cortical column circuit, which represents the internal network connection within these six layers in a single cortical column. There are two feed-forward networks connection and both of them consist of using two different cortical layers. One is L4-L2 and another L6-L5 feed-forward network interface in a single cortical column region. In this paper, the main objective is to construct the L4-L2 HTM Feed Forward network and train the network with various sequence and finally observes the behaviour of output layer L2 using HTM Classifier.

II. METHODOLOGY

The main motivation behind the HTM-Feedforward network is to let the machine perform progressed functionalities on sequence learning. So that it can rapidly learn and distinguish the repetition, a gather of similar pattern and context within the training sequence, and can predict larger sequence and perform cognitive tasks same as the human brain. Besides currently, nobody has ever tried to implement HTM Feed Forward Network for sequence learning with NeocortexApi [**13**] yet. So, L4-L2 based HTM FeedForword network is intended for this experiment.

This section is divided into three sub-sections. The proposed architecture for the construction process of the layer L4-L2 based HTM Feed-Forward Network will be described in detail in first sub-section, then the following sub-section will describe the whole algorithmic process in pseudo-code to implement the proposed HTM FeedForward Network architecture and final sub-section will explain the creating process of HTM FeedForward Network in C# code from the proposed algorithm by using NeocortexApi v.1.0.15 and Microsoft .Net Core 5.0

A. Proposed Architecture for HTM FeedForward Network

In real world each layer within the cortical column of human brain is interconnected in a way that not all the layer has Feedforward connection between each other. If we look

at the Figure 2, the cortical circuit diagram of the human brain which is suggested by NuMenta, we will notice that lower layer 6, and layer 4 has the ability to work as a receptor to take the location data input from certain object based on position and sensory data input from any human organs respectively. And layer 5 and layer 2 receive the feed forward connection from layer 6 and layer 4 respectively [1][3].

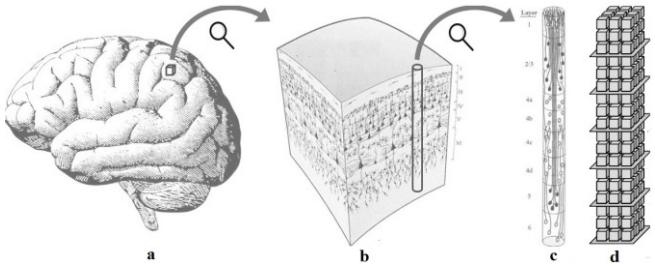

Figure 1: *a) One portion of cortical sheet from NeoCortex b) Cortical column c) A mini column with six cortical layers L1-L6. d) Single cortical column close view with group of mini columns having multiple cells in each layer*. **[14]**

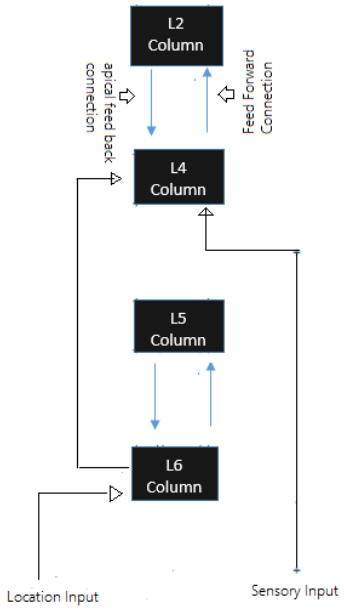

Figure 2: *Cortical circuit diagram. Which shows the internal connection between each layers.*

Figure 2 shows how each layer of the cortical column is linked. The key idea for designing the L4-L2 FeedForward network model came from studying the cortical circuit diagram. So, in Figure 3, the proposed architecture model depicts how the L4-L2 based HTM Feedforward Network model will operate with NeocortexApi. The Input Sequence Data Stream is suggested to feed to the Encoder in this architecture model, as human organs do this encoding process in real-life from actual input.Since sensory data input often goes to layer L4 in the brain (as seen in Figure 2), the proposed model also suggests that the encoder output would go to L4 SP. But now SP in the layer L4 will be in new baby born stage[**10**][**15**], which is mean the earlier stage of learning completely new things from the environment. At this stage, the use of the Homeostatic Plasticity Controller boosting algorithm in both cortical layers L4 and L2 is proposed in the architecture. This algorithm is needed because it has already experimented that, without this HPC boosting algorithm SP train will not be able to reach at stable state [**10**]. The term "***stable state***", refers the fact that the SP output of the cortex layers which is known commonly known as Sparse Distributed Representation (SDR) for active columns will remain consistent all the time for each particular input data over the iteration during training. So, in the model, this HPC boosting process will continue firstly at the SP of layer L4 until it reaches at stable state. Whenever SP of L4 has reached to stable state it will proceed to the next approach where SP training of layer L2 will start receiving stable Active cell SDRs from lower layer L4 as there is a FeedForward connection between them in the cortical layer. Since SP of L2 is in the now baby born stage now, so again HPC boosting algorithm will function here to make SP of L2 stable.

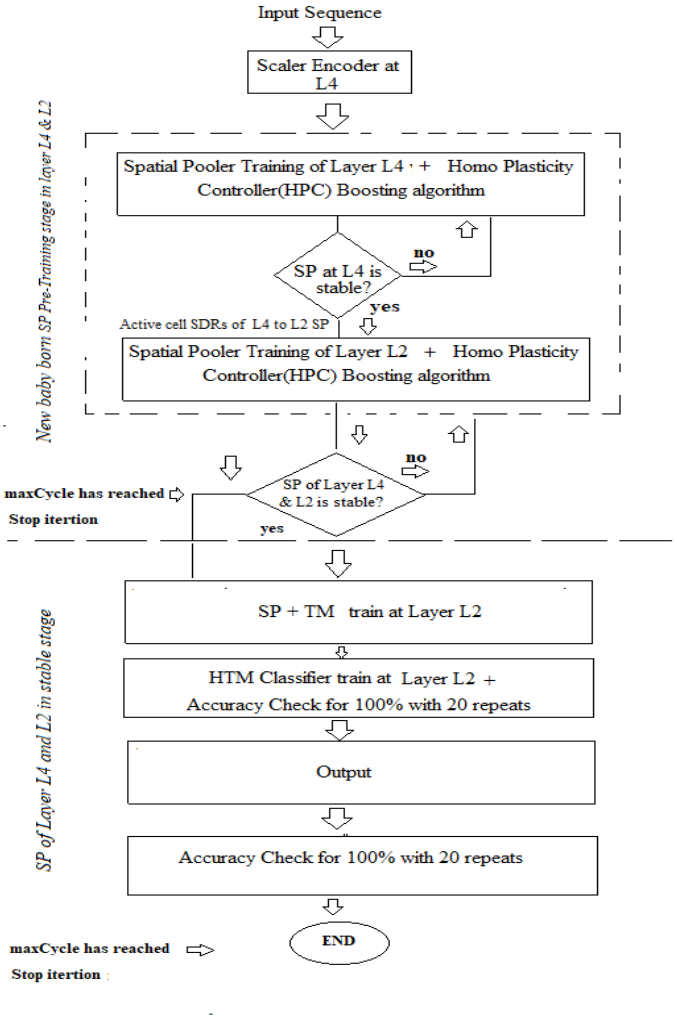

Figure 3: *Proposed Architecture model for layer L4-L2 based HTM Feed Forward Network with NeocortexApi.*

Whenever stability of SP in both layer L4 and L2 has reached the SP training process is stopped since SP in both layers is no more in the baby born stage after having stability. Then the SP+TM training process will be started at layer L2 since it is the outer layer in a cortical column of the human brain and finally with the help of HTM Classifier at layer L2 output will be generated by checking predictions accuracy of the already learned sequence patterns to

ensure that a machine is giving the output with perfection and exactness .

B. Implementation Algorithm based on the proposed architeure model at section A :

Algorithm 1: L4-L2 Based HTM FFN - Pseudo code.

```
0   function FeedForward Network Experiment ()
1|  htmConfig_L4, htmConfig_L2  // Set all parameters
2|     // for HTM configuration initialization in both layers
3|  settings, encoder ← create(settings) // Set all
4|     // parameters for creating  encoder
5|  inputValues              // Set  List to hold input sequence
6|  layerL4,layerL2,memL4,memL2 // Set all parameters
7|                  //for both  layers  and  creating connections
8|  isSP4Stable = false, isSP2Stable = false
9|  cls // Set parameters for HTM Classifier
10| sp4,sp2,tm4,tm2 // spatial pooler and temporal memory
11|                        // initialization for both layer L4 and L2
12| hpa_sp_L4, hpa_sp_l2 // Set all parameters for HPC
13|              // boosting algorithm in both layers L4 & L2.
14| //new baby born pre training stage for SP4 & SP2

15| FOR   i = 0 ; i < maxCycles
16|   FOREACH i IN inputs Set
17|     START SP4 pre-training at layer L4 by calling
18|       layerL4.Compute() method
19|         IF isSP4Stable = = true
20|     CHECK active cell SDRs is similar from L4
21|     START SP2 pre-training at layer L2 by calling
22|     layerL2.Compute(StableL4ActiveCellSDRs,
23|     true) method
24|         IF (isSP2 Stable && isSP4Stable) == true
25|          // exited from new born stage since both layer
26|           // L4 and L2 SP is in stable state
27|         ENDIF
28|   END FOREACH
29| END FOR
30| // SP + TM Process at layer L2
31| FOR   i = 0 ; i < maxCycles
32|   FOREACH i IN inputs Set
33|     START SP + TM training process at layer L2
34|     layerL2.Compute( L4ActiveCellSDRs, true)
35|                  as ComputeCycle
36|     // HTM Classifier train stage at L2
37|     key ← Get Key(previous input)
38|     cls.learn (key, array of active cells )
39|         IF key = = last predicted value by the machine
40|             COUNT matches
41|         ENDIF
42|    END FOREACH
43|      // accuracy check stage
44|      accuracy = matches / (inputs length size * 100)
45|      COUNT only the pattern with 100% accuracy
46|         IF  100% accuracy has reached 20 times
47|         // stop SP+ TM process & exited
48|         ENDIF
49| END FOR
50| END
```

The reference between the proposed architecture model & the designed Algorithm 1

- First of all an Encoder is needed to encode input data. So in pseudocode line 3 in Algorithm1 set all parameters to initiate the encoder.
- Since SP in both layers at the new baby born stage now in the proposed architecture model of L4-L2 based HTM FFN so pseudocode lines 15 to 18 denote the concept of SP pre-training at layer L4 using the HPC boosting algorithm.
- Once the SP of layer L4 has reached a stable state, the proposed model suggests starting the SP of layer L2 pre-training phase in the same way as the SP of layer L4, using the stable active cell SDRs from layer L4 to SP of L2. Pseudo code lines 19 to 23 are for this implementation at layer L2.Furthermore, the proposed model states that once the SP of layers L4 and L2 has reached a stable state, the SP pre-training phase in both layers must be stopped. As a result, lines 24 to 29 in Algorithm1 are inserted to accomplish this.
- The proposed model then recommends starting the SP+TM phase at layer L2. Lines 30 to 35 of Algorithm 1 are used for this function.
- The proposed model has used HTM classifier at layer L2 in the architecture to generate output. However, key parameters are required to initialize the HTM classifier in NeocortexApi, which carry the potential patterns based on the input data. So, Pseudo code line 37 firstly generated the key for a particular input during iteration and code line 38 denotes HTM classifier train approach at L2
- However, the proposed architecture also recommends to calculate accuracy and look for 20 times repetition of 100% accuracy so that the SP+TM process can be stooped at layer L2 during the iteration. The entire process is implemented in Algorithm1 inside code lines 39 to 50. There matches are counted during iteration for a particular input form input data set, then the accuracy is calculated at code line 44, Finally, lines 45 to 46 are examined for 100 % accuracy with 20 time repeats. Whenever it is achieved then the process is stopped and whole program exited.

C. Algorithm to Code in C# with NeocortexApi:

Algorithm 1 is translated to C# code utilizing all available features at NeocortexApi v 1.0.15 package and Microsoft .Net Core 5.0 framework. The whole project code is open-source, and you can see the current source code at this repository [**11**]. There is a file called FeedForwardExperiment.cs in which the entire process of converting the pseudocode of Algorithm 1 to C# code is performed.

I would like to split the entire code approach into eight stages to clarify the entire conversion phase of pseudo code to C# code in detail:

- Initialization stage
- SP Training Stage for SP L4 with Homeostatic Plasticity Controller Algorithm so that boosting may occur Both in baby born stage
- Check stage for Active Cell SDR of L4 is similar
- SP Training Stage for SP L2 with Homeostatic Plasticity Controller Algorithm so that boosting may occur Both in baby born stage
- Check stage L4 & L2 both are stable
- SP+TM process stage
- Classifier train stage
- Match Count stage
- Accuracy Check stage

Initialization stage:

Since pseudocode lines 1 to 12 suggest all common parameters initialization

1. So, First of all in the code I declare & initialize two variables for both layer L4 and L2 as htmConfig_L4, htmConfig_L2 respectively to configure the HTM Network for both layers.

```
HtmConfig htmConfig_L4 = new HtmConfig(new int[] { inputBits }, new int[] { numColumnsL4 })
{
    Random = new ThreadSafeRandom(42),
    CellsPerColumn = cellsPerColumnL4,
    GlobalInhibition = true,
    LocalAreaDensity = -1,
    NumActiveColumnsPerInhArea = 0.02 * numColumnsL4,
    PotentialRadius = inputBits,// Ever column is connected to 50 of 100 input cells.
    InhibitionRadius = 15,
    MaxBoost = maxBoost,
    DutyCyclePeriod = 25,
    MinPctOverlapDutyCycles = minOctOverlapCycles,
    MaxSynapsesPerSegment = (int)(0.02 * numColumnsL4),
    ActivationThreshold = 15,
    ConnectedPermanence = 0.10,
    PermanenceDecrement = 0.25,
    PermanenceIncrement = 0.15,
    PredictedSegmentDecrement = 0.1
};
```

```
// The HTM of the L2 is connected to cells of the HTM of L4.
int inputsL2 = numColumnsL4 * cellsPerColumnL4;

HtmConfig htmConfig_L2 = new HtmConfig(new int[] { inputsL2 }, new int[] { numColumnsL2 })
{
    Random = new ThreadSafeRandom(42),

    CellsPerColumn = cellsPerColumnL2,
    GlobalInhibition = true,
    LocalAreaDensity = -1,
    NumActiveColumnsPerInhArea = 0.1 * numColumnsL2,
    PotentialRadius = inputsL2, // Every columns
    InhibitionRadius = 15,
    MaxBoost = maxBoost,
    DutyCyclePeriod = 25,
    MinPctOverlapDutyCycles = minOctOverlapCycles,
    MaxSynapsesPerSegment = (int)(0.05 * numColumnsL2),
    ActivationThreshold = 15,
    ConnectedPermanence = 0.5,
    PermanenceDecrement = 0.25,
    PermanenceIncrement = 0.15,
    PredictedSegmentDecrement = 0.1
};
```

2. Declare scalar encoder :

Since during experiment have tried with numeric sequence as the input data set to encoder. So an instance of scalar encoder is needed declare & initialize before that, which can encode a Numeric (floating point) into an array of bits. In NeocortexApi some default crucial perimeters are needed to declare before initializing a scalar encoder like:

- W = Denotes the width, which is the number of bits that are set to encode a single value. It must be set to odd to avoid the centering problem
- N = Denotes number of bits at the output. Must be N >= W
- Max and Min values = Defines input range. The input sequence at lower layer must be within that range. The minimum and maximum values are termed in NeoCortex Api as “MinVal” & “MaxVal”
- Radius = Defines (W/N).I where I is the range. So we set value -1, to neglect the effect of range I

```
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);
```

3. Initialization of inputValues list with the data stream of the training sequence. This variable is needed to hold the whole training sequence during an experiment with the HTM FFN. It is configurable. We can test the Feedforward network changing input values from here. Then the RunExperiment() method is called with all of the requisite arguments to complete the remaining tasks.

```
List<double> inputValues = new List<double>(new double[] { 7, 8, 9, 10, 11, 8, 9, 12 });
RunExperiment(inputBits, htmConfig_L4, encoder, inputValues, htmConfig_L2);
```

4. The rest other parameters are initialized On Run Experiment method:

   Firstly, Booleans, Cortex layers for Connections, Temporal Memory for both L4 and L2 along with HTM Classifier are initialized

```
bool isSP4Stable = false;
bool isSP2STable = false;
var memL4 = new Connections(cfgL4); // connection at L4
var memL2 = new Connections(cfgL2); // connection at L2
layerL4 = new CortexLayer<object, object>("L4"); // init cortical layer L4
layerL2 = new CortexLayer<object, object>("L2"); // init cortical layer L2
tm4 = new TemporalMemory();// TM at L4
tm2 = new TemporalMemory();// TM at L2
//HTM Classifier initalization for layer L2
HtmClassifier<string, ComputeCycle> cls = new HtmClassifier<string, ComputeCycle>();
```

Secondly, the two parameters hpa_sp_L4 and hpa_sp_l2 are initialized for Homeostatic Plasticity Controller Algorithm and passed to Spatial Pooler for both layers so that SP train in each layer can reach to stable state from new born stage

```
//HomeostaticPlasticityController(hpc) algorithm hpa_sp_L4,hpa_sp_L2
SpatialPooler sp4 = new SpatialPooler(hpa_sp_L4); // SP at L4 with hpc
SpatialPooler sp2 = new SpatialPooler(hpa_sp_L2); // SP at L2 with hpc
sp4.Init(memL4);
sp2.Init(memL2);
```

SP L4 training stage:

In this stage, pseudocode lines 15 to 20 of Algorithm 1 are translated in in C# code. Here SP of layer 4 will quickly be trained with the input data from Sequence Stream iterating per cycle and boosting occurs with the help of the HPC Algorithm in NeocortexApi until SP of L4 doesn't reach to stable state.

```
for (int i = 0; i < maxCycles; i++)
{
    matches = 0;
    cycle = i;
    Debug.WriteLine($"-------------- Newborn Cycle {cycle} at L4 SP region  ----------

    foreach (var input in inputs)
    {
        Debug.WriteLine($" INPUT: '{input}'\t Cycle:{cycle}");
        Debug.Write("L4: ");
        var lyrOut = layerL4.Compute(input, learn);
        var activeColumns = layerL4.GetResult("sp") as int[];
        int[] cellSdrL4Indexes = memL4.ActiveCells.Select(c => c.Index).ToArray();
```

Check stage for Active Cell SDR of L4 is Similar:

In neocortexapi there is an issue that active cell sdr indexes of layer L4 don't remain consistent all-time for each input after reaching to a stable state. Which makes SP of L2 training unstable later. So, in the Algorithm 1 pseudo code line 20 is added to check the active cell SDRs similarity of layer L4 after it has reached to stable state before passing to SP at layer L2.So a Dictionary is maintained for active cell SDR Log in the code after SP of layer L4 has reached to a stable state for each particular input for the first time. Then a check operation is performed to make sure active cell SDRs from L4 is similar comparing with active cell SDRs log in the very next cycle of the iteration for each particular input.

```
Array.Sort(cellSdrL4Indexes);
if (!L4_ActiveCell_sdr_log.ContainsKey(input))
{
    L4_ActiveCell_sdr_log.Add(input, cellSdrL4Indexes);
}
else
{

    if (L4_ActiveCell_sdr_log[input].SequenceEqual(cellSdrL4Indexes))
    {
        Debug.WriteLine($"Layer4.Compute() is giving similar cell sdr indexes for input : {input} aft
        isSimilar_L4_active_cell_sdr = true;
    }
    else
    {
        isSimilar_L4_active_cell_sdr = false;
        Debug.WriteLine($"Layer4.Compute() is giving different cell sdr indexes for input : {input} a
        Debug.WriteLine($"Sdr Mismatch with L4_ActiveCell_sdr_log after reaching to stable state");
        Debug.WriteLine($" L4_ActiveCell_sdr_log output for input {input}: { Helpers.StringifyVector(
        Debug.WriteLine($"L4 out sdr input:{input} {Helpers.StringifyVector(cellSdrL4Indexes)}");

    }
}
```

If similar sdr indexes are not received in such case the cellL4indexes are modified with the sdr indexes from sdr log, which ensures SP of L2 will get a similar active cell sdr pattern during training.

```
if (!isSimilar_L4_active_cell_sdr)
{

    cellSdrL4Indexes = L4_ActiveCell_sdr_log[input];
}
```

SP Training Stage for SP L2:

At this stage, pseudocode lines 21 to 23 are converted in C# code. Whenever SP of layer 4 has reached to table state then the training process for SP of Layer 2 has started with the receiving active cell sdr indexes from layer4

```
InitArray(inpCellsL4ToL2, 0);
ArrayUtils.SetIndexesTo(inpCellsL4ToL2, cellSdrL4Indexes, 1);
Debug.WriteLine($"L4 cell sdr to L2 SP Train for Input {input}: ");
layerL2.Compute(inpCellsL4ToL2, true);
```

Check stage for L4 & L2 both are stable:

The cycle will iteration will break if both layers has reached to stable state. As SP in both layers is not more in new born stage. Pseudo code line 24 to 29 indicate stage which is implemented in the code as:

```
if (isSP4Stable && isSP2STable)
    break;
```

SP+TM train for L2:

When both layers have reached a stable state, the SP+TM process will begin for layer L2 with the active cell Sdr Indexes from layer L4 for each particular input during each cycle, as shown in pseudocode lines 30 to 35. So in the source code, it is achieved in this way

```
ArrayUtils.SetIndexesTo(inpCellsL4ToL2, cellSdrL4Indexes, 1);
var layerL2Out = layerL2.Compute(inpCellsL4ToL2, true) as ComputeCycle;
```

HTM Classifier at L2:

To generate output from layer L2 an HTM Classifier is needed. So in the pseudo code .lines 36 to 38 denote to do this process. In NeocortexApi the learn () method in HTM classifier takes a key as an argument, which a string of possible sequence based on the current input and memorize the SDR indexes for the given key. So, the code is followed this approach to have this.

```
// key is a string of sequential patterns during each iteaeration based on input context
key = GetKey(previousInputs, input);
// active cell SDR from layer L2
if (layerL2Out.ActiveCells.Count == layerL2Out.WinnerCells.Count)
{
    actCells = layerL2Out.ActiveCells;
}
else
{
    actCells = layerL2Out.WinnerCells;
}
/// <summary>
/// HTM Classifier at  Layer 2
/// </summary>
cls.Learn(key, actCells.ToArray());
```

Match count Stage:

In pseudo code lines 39 to 40 are added so that if the key value is equal with the last Predicted Value then match variable will be incremented. So in code, it is like the following one

```
if (key == lastPredictedValue)
{
    matches++;
    Debug.WriteLine($"Match.
}
else
    Debug.WriteLine($"Missmatch!
```

Accuracy Check Stage:

This the ultimate stage which is denoted within the pseudo code line 43 to 50. During this stage, 100% accuracy is being examined and counted what number of time this accuracy has received. This is done to ensure perfection in the prediction accuracy so that a machine will either give the accurate prediction result after learning or not. So, If 100% accuracy is reached with 20 repeats then it can be said that the machine is perfectly get trained with the input sequence, so the whole program is stopped then. This whole process on the C# code as

```
// Calculate accuracy
double accuracy = (double)matches / (double)inputs.Length * 100.0;
// count number of 100% accuracy
if (accuracy >= 100.0)
{
    maxMatchCnt++;

    // Experiment is completed if we are 20 cycles long at the 100% accuracy.
    if (maxMatchCnt >= 20)
    {
        learn = false;
        // Program exited  after having 100% accuracy more than 20 times
        break;
    }
}
```

This is how L4-L2 HTM FeedFroward network is implemented in C# code from the pseudocode of Algorithm 1, with NeocortexApi v.1.0.15

III. EXPERIMENT RSEULTS & FINDINGS

More than twenty experiments are done with several types of input sequences. It is observed that most of the time L2 is stable and 100% accuracy has reached. All of the unit tested sequences & results output are uploaded in github. [**6**][**12**]

Findings:

HTM Classifier Limitation

Though each time have received 100% accuracy for some sequence didn't achieve 100% accuracy. It's because have noticed a limitation in the HTM Classifier of NeocortexApi. It returns the 1st matching context instead of looking for the entire array for possible inputs. That's why mismatch happens and learning sequence doesn't reach to 100% accuracy

For example let's look at experiment result for

Training Sequence: 2-6-6-7-6-6-8-1
Though indx 5 and indx 8 both has same similarity. But HTM classifier is returning index 5 since it has watched this context earlier compare to indx8

<indx:0 inp/len: 2-6-6-7-6-6-8/50 = similarity 0 135, 294, 459, 733, 770, 818, 1293, 1337, 1919, 1920, 1955, 2020, 2117, 2434, 2776, 2924, 3124, 3237, 3578, 3696, 3754, 4191, 4240, 4308, 4442, 4731, 4826, 4991, 5045, 5176, 5277, 5329, 6355, 6777, 6864, 6940, 7182, 7408, 7575, 7682, 7891, 8041, 8065, 8179, 8319, 8810, 9249, 9378, 9736, 9892,

<indx:1 inp/len: 2-6-6-7-6-6-8-1/50 = similarity 0 10, 464, 663, 1184, 1309, 1353, 1889, 2209, 2220, 2400, 2497, 2606, 2869, 2888, 2986, 3076, 3322, 3380, 3620, 3679, 4364, 4674, 4766, 4809, 5302, 5617, 5672, 5696, 5784, 5818, 5954, 6499, 6630, 7135, 7207, 7354, 7535, 8102, 8233, 8686, 8701, 8812, 8847, 8906, 8923, 9132, 9405, 9639, 9775, 9917,

<indx:2 inp/len: 6-6-7-6-6-8-1-2/50 = similarity 0 11, 83, 349, 1073, 1191, 1439, 2015, 2228, 2444, 2675, 2730, 2870, 3022, 3424, 3444, 3522, 3719, 3801, 4090, 4215, 4238, 4291, 4777, 4797, 5021, 5199, 5239, 5583, 5641, 5798, 5900, 5998, 6040, 6185, 6355, 6417, 6421, 6612, 6759, 6979, 7283, 7450, 7472, 7639, 7841, 7919, 8117, 9025, 9406, 9623,

<indx:3 inp/len: 6-7-6-6-8-1-2-6/50 = similarity 0 30, 170, 320, 373, 426, 486, 632, 682, 913, 1028, 1099, 1576, 1764, 2190, 2212, 2567, 3217, 3272, 4298, 4656, 4692, 4750, 4900, 4962, 5237, 5291, 5454, 5512, 5613, 5760, 5882, 5920, 6109, 6143, 6287, 6306, 6502, 6524, 6645, 7156, 7427, 8502, 8527, 8605, 8942, 9055, 9394, 9510, 9573, 9927,

<indx:4 inp/len: 7-6-6-8-1-2-6-6/50 = similarity 0 24, 167, 330, 368, 428, 482, 625, 686, 901, 1035, 1090, 1560, 1776, 2183, 2203, 2577, 3211, 3278, 4284, 4654, 4683, 4756, 4912, 4975, 5229, 5296, 5456, 5504, 5616, 5776, 5895, 5937, 6113, 6141, 6290, 6307, 6517, 6530, 6640, 7158, 7439, 8500, 8537, 8600, 8940, 9049, 9390, 9502, 9579, 9920,

>indx:5 inp/len: 6-6-8-1-2-6-6-7/50 = similarity 50 603, 758, 784, 812, 975, 1690, 2129, 2358, 2585, 2631, 2937, 3054, 3244, 3733, 3766, 3790, 3975, 4317, 4882, 4992, 5128, 5435, 5474, 5535, 5625, 5961, 6329, 6766, 6869, 6916, 6947, 7033, 7217, 7703, 7817, 7878, 7986, 8030, 8173, 8497, 8555, 8899, 9218, 9251, 9308, 9719, 9848, 9863, 9896, 9943,

<indx:6 inp/len: 6-8-1-2-6-6-7-6/50 = similarity 0 23, 175, 335, 364, 436, 498, 639, 681, 911, 1027, 1098, 1570, 1771, 2199, 2218, 2568, 3218, 3276, 4296, 4643, 4681, 4758, 4904, 4967, 5236, 5290, 5448, 5515, 5603, 5779, 5886, 5938, 6116, 6142, 6293, 6305, 6507, 6525, 6651, 7149, 7435, 8504, 8533, 8619, 8956, 9043, 9396, 9504, 9561, 9938,

<indx:7 inp/len: 8-1-2-6-6-7-6-6/50 = similarity 0 39, 161, 326, 376, 430, 484, 627, 680, 902, 1022, 1085, 1574, 1770, 2181, 2213, 2573, 3206, 3275, 4299, 4652, 4687, 4752, 4907, 4970, 5232, 5288, 5441, 5506, 5611, 5764, 5885, 5926, 6112, 6145, 6283, 6319, 6514, 6521, 6655, 7152, 7434, 8518, 8529, 8607, 8951, 9047, 9387, 9506, 9572, 9934,

<indx:8 inp/len: 1-2-6-6-7-6-6-8/50 = similarity 50 129, 281, 455, 734, 760, 812, 1297, 1335, 1908, 1938, 1959, 2026, 2114, 2430, 2760, 2937, 3127, 3238, 3561, 3691, 3752, 4180, 4242, 4317, 4443, 4730, 4833, 4992, 5051, 5169, 5268, 5338, 6359, 6766, 6869, 6947, 7193, 7419, 7563, 7699, 7895, 8056, 8072, 8173, 8307, 8815, 9251, 9376, 9737, 9896,
Current Input: 6 | New Predicted Input Sequence: 6-6-8-1-2-6-6-7

-------------- 8 ---------------
PPPPPPPPPPPP
Active segments: 0, Matching segments: 10
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP
Active segments: 50, Matching segments: 50
Mismatch! Actual Sequence: 1-2-6-6-7-6-6-8 - Last Predicted Sequence: 6-6-8-1-2-6-6-7

Occasional Instability at TM output at layer L4:

During the experiment we have noticed that whenever SP of layerL4 has reached to stable state by executing the method layerL4.Compute() then `cellSdrL4Indexes` which is the TM output of layer L4 commonly known as indexes of ActiveCellsSDR does not remain similar all time rather it is giving completely different sdr indexes **[8]**, for each particular input from the data set during iteration. This occurrence makes almost impossible for SP of layer L2 to reach at stable state since cell sdrs varies at L2 SP input from layer L4 during training.

```
// check SP of L4 is in stable state
If(isSP4Stable){
var lyrOut = layerL4.Compute(input,true);
//active cell sdr indexes fluctuate here
//due to this SP of L2 doesn't reach to stable
state
int[] cellSdrL4Indexes =  memL4.ActiveCells.Selec
t(c => c.Index).ToArray();
. . .
}
```

However, why the TM doesn't give us stable active cell sdr for any types of HTM Network Configurations, the exact reason is not founded yet.

But interesting thing is that, this issue does not happen all time for all types of network Configurations. If tuning is performed for some of the HTM network configurations parameter in both layers L4 and L2 such makes L4 TM stable by giving always similar active cell sdr for a input at output whenever SP of layer L4 has reached to a stable state. Tuning actually mean training the L4-L2 Based HTM Feedforward network with various types input sequences varying or changing the HTM network configuration parameters then observed the experiment result. Have found some specific network configurations which will solve this issue also work perfectly to make SP of layer L2 stable each time.

Table 1: HTM Configuration Parameters Tuning effects on L4-L2 based HTM FFN

Tuning HTM network Configuration Parameters Values at L4	Tuning HTM network Configuration Parameters Values at L2	SP L4	TM L4	SP L2
numColumns4 = 2048, Activation Threshold=15, Connected Permanence = 0.15, NumActive ColumnsPerInh Area = 0.2 * numColumnsL4	numColumns2 = 500, Activation Threshold=10, Connected Permanence = 0.10, NumActive ColumnsPerInh Area = 0.02 * numColumnsL2	Reach to Stable Sate	Doesn't Reach to Stable State	Doesn't Reach to Stable State
numColumns4 = 1024, Activation Threshold=15, Connected Permanence = 0.10, NumActive ColumnsPerInh Area = 0.02 * numColumnsL4	numColumns2 = 1024, ActivationThreshold=15, Connected' Permanence = 0.10, NumActive ColumnsPerInh Area = 0.02* numColumnsL2	Reach to Stable Sate	Doesn't Reach to Stable State	Doesn't Reach to Stable State
numColumns4 = 500, Activation Threshold=15, Connected Permanence = 0.10, NumActive ColumnsPerInh Area = 0.02 * numColumnsL4	numColumns2 = 500, ActivationThreshold=15, ConnectedPermanence = 0.5, NumActive ColumnsPerInh Area = 0.1* numColumnsL2	Reach to Stable Sate	Reach to Stable Sate	Reach to Stable Sate
numColumns4 = 1024, Activation Threshold=15, Connected Permanence = 0.10, NumActive ColumnsPerInh Area = 0.02 * numColumnsL4	numColumns2 = 500, Activation Threshold=15, Connected Permanence = 0.5, NumActive ColumnsPerInh Area = 0.02* numColumnsL2	Reach to Stable Sate	Reach to Stable Sate	Reach to Stable Sate

Table 1, holds our experiment summary of tuning network Configuration Parameters at both layers and its effect on the layer L4-L2 Based HTM Feed Forward Network with neocortexapi

IV. OPEN ISSUE

In Figure 2, though there is feed forward connection from layer L4 to L2 but there is also the presence of reverse feedback connection from layer L2 to L4. In neuroscience this reverse connection known as the Apical Feedback connection. NuMenta implemented this reverse FeedForward connection from layer L2 to L4 in Python [**9**]. They suggested a new TM algorithm for that in lower layer L4 to perform with apical feedback connection from upper layer L2. They name it as Apical Tiebreak TM [**5**] [**7**]. But how this Apical Tiebreak TM algorithm works we didn't find any explanation or papers on that. Again in the current version of NeocortexApi there is no implementation for the Apical Tiebreak TM algorithm. So it is an open issue for in future to implement Apical Tie Break TM for NeocortexApi and check L2-L4 based HTM FFN behaviour with that

V. CONCLUSION

HTM FeedForward network is a multilayer based artificial network arrangement inspired by the cortical circuit connection between the six layers in a single cortical column in the NeoCortex of the human brain. Within the cortical circuit, Feedforward connection between layer lower L4 to upper L2 plays the vital role to perform cognitive learning and predicting task in a human brain taking sensory data as input by various human organs from the environment to lower layer L4. So, being influenced by the working principle of the L4-L2 network in a human brain. This paper mainly focuses on implementing an artificial L4-L2 based HTM Feed Forward Network and training process of the implemented network with sequential data using HTM Classifier at output layer L2 so that a machine can learn and predict just like a human brain does. To achieve this we propose an architecture and an algorithm based on the proposed model of the architecture for constructing L4-L2 based HTM FFN using the latest available features in neocortexapi, which is open source implementation of HTM Cortical learning algorithm. This paper also included a C# code solution based on the proposed algorithm, as well as a brief description of the entire process of using neocortexapi to accomplish this. The result shows that the constructed layer L4-L2 based HTM FFN with layer L4-L2 based HTM FFN can learn the transition in the sequence on the upper layer.

REFERENCES

[1] https://www.frontiersin.org/articles/10.3389/fncir.2017.00081/full

[2] https://www.researchgate.net/publication/309778443_A_comparative_study_of_HTM_and_other_neural_network_models_for_online_sequence_learning_with_streaming_data

[3] https://numenta.org/resources/HTM_CorticalLearningAlgorithms.pdf

[4] https://numenta.com/neuroscience-research/research-publications/papers/a-framework-for-intelligence-and-cortical-function-based-on-grid-cells-in-the-neocortex/

[5] https://github.com/numenta/htmresearch/blob/master/htmresearch/algorithms/apical_tiebreak_temporal_memory.py

[6] https://github.com/MukitCSTE/HTMFFN/tree/main/Unit%20Test%20Experiment%20Results-L2Stable-Accurecy100%25

[7] https://discourse.numenta.org/t/a-theory-of-how-columns-in-the-neocortex-enable-learning-the-structure-of-the-world/2524/100

[8] https://gist.github.com/MukitCSTE/e3cfb9c119cbd5eb6bd996a8fb1970ef

[9] https://github.com/numenta/htmresearch/blob/master/htmresearch/frameworks/layers/l2_l4_network_creation.py

[10] https://github.com/ddobric/neocortexapi/blob/master/NeoCortexApi/Documentation/Experiments/ICPRAM_2021_76_CR.pdf

[11] https://github.com/MukitCSTE/HTMFFN/tree/main/ML2021-5.2%20HTM%20FeedForward%20network

[12] https://github.com/MukitCSTE/HTMFFN/tree/main/Unit%20Test%20Project/FeedForwardNetExperimentUnitTest

[13] https://github.com/ddobric/neocortexapi

[14] https://discourse.numenta.org/t/mini-macro-and-cortical-column/7575

[15] Turrigiano, Nelson. (2004). Homeostatic plasticity in the developing nervous system. Nature Reviews Neuroscience, 97–107