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

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Implementation of CLA Classifier

Anik Saha	Kazi Md Wahidul Gani	Meem Shahrin	Mou Saha	Riyad Ul Islam	Sami Patwary
1325658	1324390	1325603	1327931	1324662	1324646

Abstract— HTM CLA is a new method of machine learning, a cognitive approach that is scientifically influenced, based on the concepts of how human brain functions. CLAs are an effort by Numenta Inc. to generate a computational prototype of perceptual analysis and learning inspired by the neocortex in the human brain. The approach welcomes the hierarchical structure and suggests a system for memory prediction of what will happen in the near future. This paper presents a detailed description of HTM's Cortical Learning Algorithm (CLA). The algorithm is implemented using three classes in dotnet Core 3.0 platform. The results are observed for different iterations and inputs in the classifier.

Keywords— Hierarchical temporal memory (HTM), Cortical Learning Algorithms (CLA), SDR, NuPIC

a. Introduction

Over the past decade, machine learning and AI have succeeded at performing many tasks that were unimaginable earlier. The manner in which these systems execute these tasks is fundamentally different from how our brain performs the same tasks. Hierarchical Temporal Memory (HTM) is a recent innovation in cognition science. Developed in 2005 by Numenta Inc., an artificial intelligence research firm in the US, HTMs attempt to capture the way the human brain learns and infers its environment [1].

The job of the HTM algorithms is to learn the temporal sequences from a stream of input data, i.e. to build a model of which patterns follow which other patterns. This job is difficult because it may not know when sequences start and end, there may be overlapping sequences occurring at the same time, learning has to occur continuously, and learning has to occur in the presence of noise. Learning and recognizing sequences is the basis of forming predictions. Once an HTM learns what patterns are likely to follow other patterns, it can predict the likely next pattern(s) given the current input and immediately past inputs [2].

The goal for each HTM region is to learn patterns from the input data while accounting for spatial and temporal variability. This is achieved with an algorithm called Cortical Learning Algorithm. HTM-CLA is based on the structure and function of the human neocortex which works as a memory-based prediction system. It offers the groundwork for building machines that approach or exceed human level performance for many cognitive tasks. In this paper we evaluate our implantation of CLA classifier using dotnet Core 3.0 platform. It also elaborates on the obtained result of the implementation.

b. Methodology

The HTM-CLA Classifier is not biologically inspired, but is a useful classifier for predicting the SDR output from the temporal memory and generating predictions. HTM-CLA is a theory of how the human brain works. There are three important attributes of brain in designing HTM classifier. These are-

- Firstly, the brain is naturally a hierarchical system. In this hierarchy, there is flow of signal in both direction and within the region itself.
- Secondly, the human brain naturally store all the information in temporal. In every situation of the brain learning, time is a universal feature.
- Thirdly, the human brain is basically a memory system. All the brain cell and their connection are in a manner that can store the patterns over time if it tries to memorize and predict patterns over time.

The implementation of HTM-CLA is available in the NuPIC framework [1],[2]. Figure 2 shows the process flow of the NuPIC framework. Input data are first sent to encoders so that it can convert them into binary bits. The spatial pooler compute the list of columns that win due to the bottom-up input at fixed time. And then it send the list as input to the temporal pooler. Then the temporal pooler enables the cells of a column to learn to represent the input in the context of time. Classifier tries to infer the output from active columns in the upper region in the hierarchy.

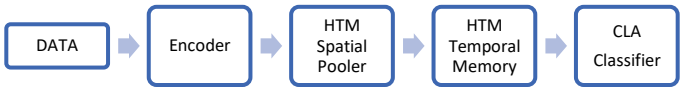

Figure 1: HTM-CLA Hierarchy

Algorithm

CLA classifier attempts to learn a function of an SDR at time t (SDR_t) , such that it produces a probability distribution over the predicted field (PF), k steps into the future [4]:

$$f(SDR_t) \longrightarrow P(PF_{t+k})$$

The CLA Classifier takes the following parameters:

- alpha: The alpha is used to calculate the running averages. A lower alpha results a longer memory.

- steps: The set of different steps for multi-steps predictions to learn, e.g. (1,3,7,12).

To do this, for each predicted step (*k*) , the CLA Classifier maintains a mapping of:

$$f(SDR_{t-k}) \longrightarrow PF_t$$

This given mapping saves a track of input SDRs it has seen, so, given an input, it can refer to the history and set the probability distribution over the PF from a given input. It does this by:

- If we are predicting a categorical value, then we do not consider the running average array.
- Saving H and A arrays with the shape of N×B, where N is the number of bits in the SDR and B is the bucket numbers on the PF as defined by the input encoding:
- H: A histogram that saves the relative frequency of bucketed input values when its corresponding SDR bit (n) is active. That is:

$$H[n][b] = \frac{times\ bucketed\ input\ was\ seen\ when\ n\ was\ active}{times\ n\ was\ active}$$

- A: A running average of the input values, whose length is set by alpha. When this array's corresponding SDR bit n is active with a given predicted field value v that falls into bucket b, the array is updated by:

$$A[n][b] = ((1-\alpha) \times A[n][b]) + \alpha \times v$$

So, when a bucket covers a range of values (i.e. non-categorical values), we don't get a prediction about a specific range, rather, we get average value that fell into that bucket.

- For a particular input SDR of length *N* with *N'* active bits, predictions are created for each bucket (b) of the predicted field, at each timestep (k) by averaging the product of the associated histogram value and running average table for each active bit:

$$P(PF_{t+k}) = \left[\frac{1}{N'} \sum_{SDR[n]=1} A[n][b] \times H[n][b] : b \in [1, B]\right]$$

So, for each bucket of the predicted field we will get the probability, which could be very low for all buckets. We can use the bucket that has highest probability as our prediction, or not, depending on the context and the intension of the prediction. For example, the highest prediction may be for 100% engine load with a 0.1% probability, such a low probability would not necessitate the same response that a 95% probability would with the same load.

Our Implementation:

We have implemented our algorithm in .net core 3.0 in C#. Here We implemented three classes named **CLAClassifier**, **BitHistory** and **ConversionExtensions** **.**

Class CLAClassifier:

I. Field Description

Parameters:

- ***alpha***(double): The alpha used to compute running averages of the bucket duty cycles for each activation pattern bit. A lower alpha result in longer term memory.
- ***learnIteration*** (int): The bit's learning iteration. This is updated each time store() gets called on this bit.
- ***recordNumMinusLearnIteration*****(**int**)**: This contains the offset between the recordNum (provided by caller) and learnIteration (internal only, always starts at 0).
- ***maxBucketIndex*****(**int**)**: This contains the value of the highest bucket index we've ever seen It is used to pre-allocate fixed size arrays that hold the weights of each bucket index during inference
- ***steps(***IList<int>) : The sequence different steps of multi-step predictions
- ***patternNZHistory*****(**IList<Tuple<int, int[]>>**):** History of the last _maxSteps activation patterns. We need to keep these so that we can associate the current iteration's classification with the activationPattern from N steps ago
- ***activeBitHistory*****(**Dictionary<Tuple<int, int>**):** These are the bit histories. Each one is a BitHistory instance, stored in this dict, where the key is (bit, nSteps). The 'bit' is the index of the bit in the activation pattern and nSteps is the number of steps of prediction desired for that bit
- ***actualValues***(IList<T>): This keeps track of the actual value to use for each bucket index. We start with 1 bucket, no actual value so that the first infer has something to return.

II. Constructor Details

1. *public CLAClassifier()*

CLAClassifier no-arg constructor with defaults. Here default values are- {1} array for steps, {0.001} for alpha, {0.3} for actValueAlpha

2. *public CLACLassifier(IList<int> steps,*
   double alpha,
   double actValueAlpha)

Constructor for the CLA classifier.

Parameters:
steps - order of the different steps of multi-step predictions for learning
alpha - The alpha used to compute running averages of the bucket duty cycles for each activation pattern bit. A lower alpha result in longer term memory.

III. Method Detail

Compute

```
Classification<T>Compute(int recordNum,

Dictionary<string, object> classification,  int[]
patternNZ, bool learn,bool infer)
```

Process one input sample. This method can be called from outside and here inputs and outputs aren't fixed size.
Parameters:

- ***recordNum*** - Record number of this input pattern and it is incremented by 1 unless there are missing records. So that we don't get confused by missing records.
- ***classification*** **-** Map of the classification information: ***bucketIdx*****:** index of the encoder bucket, ***actValue:*** actual value going into the encoder
- ***patternNZ*** **-** list of the active indices from the output
- ***learn*** **-** if true, learn sample
- ***infer*** **-** if true, perform inference

Returns:
Classification have inference results and there is one entry for each step in steps, where the key is the number of steps, and the value is an array having the relative probabilities for each bucketIdx starting from 0. There is also parameter average which calculate average actual value to use for each bucket. The key is 'actualValues'. for example: { 1 : [0.1, 0.3, 0.2, 0.7], 4 : [0.2, 0.4, 0.3, 0.5], 'actualValues': [1.5, 3.5, 5.5, 7.6] }
Algorithms:
Here is the algorithm of updating moving average of actual values.
If the value is scalar value:

```
double val = ((1.0 - actValueAlpha) *
Convert.ToDouble(actualValues[bucketIdx])) +
                        (actValueAlpha *
(Convert.ToDouble(actValue)));
actualValues[bucketIdx] =
ConversionExtensions.Convert<T>(val);
```

If it is not scalar, then the value is category value and then each bucket only have one value. Then we need not to calculate moving average. Here this is the algorithm of training each pattern which is already saved in our history: pattern which is already saved in our history:
Here this is the algorithm of training each pattern which is already saved in our history.

```
actualValues[bucketIdx] = (T)actValue;
```

Here this is the algorithm of training each pattern which is already saved in our history.

```
foreach (var n in steps)
            {
               nSteps = n;
               // Do we have the pattern that should be assigned
to this classification
               // in our pattern history? If not, skip it
               bool found = false;
               foreach (var t in patternNZHistory)
               {
                  iteration = t.Item1;
                  learnPatternNZ = t.Item2;
                  if (iteration == learnIteration - nSteps)
                  {
                     found = true;
                     break;
                  }
                  iteration++;
               }
               if (!found) continue;
               // Store classification info for each active
bit from the pattern
               // that we got nSteps time steps ago.
                foreach (int bit in learnPatternNZ)
               {
                  // Get the history structure for this bit
and step
                  var key = Tuple.Create(bit, nSteps);
                  BitHistory history = null;
                  activeBitHistory.TryGetValue(key, out
history);
                  if (history == null)
                  {
                     history = new BitHistory(alpha);
                     activeBitHistory.Add(key, history);


                  }
                  history.store(learnIteration, bucketIdx);
               }
         }
```

Class BitHistory

I. Field Description

- ***alpha*****(**double**):**Store reference of alpha of CLAClassifier class.
- ***stats(***Dictionary<int,double>**):** Dictionary of bucket entries. The key is the bucket index, the value is the dutyCycle, which is the rolling average of the duty cycle
- ***lastTotalUpdate(****int****):*** lastUpdate is the iteration number of the last time it was updated.

II. Constructor Details

```
Public BitHistory(double alpha)
```

Constructs a new BitHistory
Parameters:
alpha- alpha value of the CLAClassifier Class instance.

III. Method Details

store

```
public void store(int iteration, int bucketIdx)
```

Saving a new item in history of CLAClassifier. This method gets called for a bit when it is active and learning parameter is enabled.
Storing duty cycle by normalizing it to the same iteration as the rest of the duty cycles which is lastTotalUpdate.
The duty cycle is needed to calculate the current iteration only at inference and only when one of the duty cycles gets too large. When all the duty cycles are at the same iteration their ratio is the same as it would be for any other iteration, because the update is a multiplication by a scalar that depends on the number of steps between the last update of the duty cycle and the current iteration.
Parameters:
iteration - the learning iteration number
bucketIdx - the bucket index to store
infer

```
public void infer(double[] votes)
```

Look up and return the votes for each bucketIdx for this bit.
Parameters:
votes - array, first initialized with all 0's which later filled in with the votes for each bucket. The vote for bucket index I set value in votes[I].

c. Results

All of the test cases are implemented in **ClaClassifierTest** class.

I. Multiple Iteration with single input in CLA classifier:

For input bucket index = 0 while active pattern is [1,5] and with ten iteration in CLA classifier.

Table 1. Growing probability distribution of bucket 0

Iteration	Probability of Bucket 0	Probability of Bucket 1
0	1	
1	0	1
2	0.33333	0.66667
3	0.39999	0.60000
4	0.42857	0.57143
5	0.44444	0.55556
6	0.45454	0.54546
7	0.46154	0.53846
8	0.46666	0.53333
9	0.47059	0.52941

In CLA classifier, it doesn't learn the data in the first iteration. Main thing observed here is the number of iterations has been examined, where the greater number of iterations bring better output. With every iteration, the learning grows in returns increases the probability distribution. This test case is implemented using **TestWithDiffertBucketIndexWithMultipleIteration** method.

II. Multiple Inputs with 5 Iterations at CLA Classifier:

For the first iteration, considered Bucket Index = 4 and actual value of 34.7 is given from the encoder.

So, predicted probability when the active pattern is [1, 5, 9]

Table 2. Probability distribution

Bucket	Probability
0	1

As per CLA Classifier algorithm, classifier doesn't learn data in the first iteration.

Now in the second iteration, Bucket Index is 5 with the actual value of 41.7 from the encoder is taken into consideration.

With activated pattern [0, 6, 9, 11], the predicted probability is examined-

Table 3. Probability distribution

Bucket	Probability
0	0.2
1	0.2
2	0.2
3	0.2
4	0.2

Again, for Bucket index = 5 and actual value = 44.9 taken from encoder for the third iteration.

Where [6, 9] is considered as active pattern.

Table 4. Probability distribution

Bucket	Probability
0	0
1	0
2	0
3	0
4	0
5	1

While considering fourth iteration, actual value = 42.9 is given from encoder and setting bucket Index = 4. Active pattern is [1, 5, 9]

Table 5. Probability distribution

Bucket	Probability
0	0
1	0
2	0
3	0
4	0
5	1

In the last iteration, Bucket Index = 4 and actual value = 34.7 when the activated pattern [1, 5, 9]

Table 6. Probability distribution

Bucket	Probability
0	0
1	0
2	0
3	0
4	0.12300123
5	0.87699877

Though we set actual value for bucket index 4 are (34.7, 42.9, 34.7) and bucket index 5 are (41.7, 44.9), actual Value for Bucket index 4 is 35.52 and for bucket index 5 is 42.02 have been found. As in the algorithm for actual value we set

Actual value for current bucket index = ((1.0 - actValueAlpha) * previous actual value of same bucket index) + (actValueAlpha * actual value of current bucket index given).

This test case is implemented using testComputeComplex method.

Limitations:

- In this CLA classifier algorithm actual value can't be (-1), as (-1) defined as null.
- By overlapping the iteration classifier won't work.

d. Conclusion

Neocortex is a high activity of brain such as sensory perception and generating of next thought in human brain. Similarly, CLA classifier is calculating next interpretation by using a function of SDR. The HTM cortical learning algorithm embodies what we believe is a basic building block of neural organization in the neocortex. It shows how a layer of horizontally connected neurons learns sequences of sparse distributed representations. In this paper, we performed three classes: CLAClassifier, BitHistory and ConversionExtensions. Implementation by those classes, we can improve CLA classifier's performance and prediction in program. Although after the integration of SDR classifier to NuPIC, and it's better performance on example data, now it's disputing that SDR classifier to be the default classifier for CLA classifier region.

e. References

1) Numenta, Hierarchical Temporal Memory including HTM Cortical Learning Algorithms, HTM CLA white paper - VERSION 0.2.1, SEPTEMBER 12, 2011
2) G. K. F. Z. Jahan Balasubramaniam, "Enhancement of Classifiers in HTM-CLA Using Similarity Evaluation Methods," in *19th International Conference on Knowledge Based and Intelligent Information and Engineering*, 2015.
3) M. S. Abdullah Alshaikh, "A Cortical Learning Movement Classification Algorithm for Video Surveillance," *International Journal of Computer Applications Technology and Research*, vol. 6, no. 12, 2017.
4) "CLA Classifier," *HTM Forum*, 20-Nov-2019. [Online]. Available: https://discourse.numenta.org/t/cla-classifier/2143. [Accessed: 10-Apr-2020].