Software Engineering

By Damric Dobric/Andreas Pech

Migration of video learning project

Nusrat Jahan Sumi
Matriculation ID: 1345476

Mashnunul Huq
Matriculation ID: 1384042

***Abstract*: This paper represents an improving machining learning algorithm Hierarchical Temporal Memory (HTM) which is using Spatial pooler for learning Video data. The SP model shows how neurons learn by feedforward connections and form effective classification of the input frame. It converts binary input pattern into space distributed representation (SDR) by using Cortical Learning rules and homeostatic plasticity control for frame pattern prediction. The result of the learning is tested by giving the trained model an arbitrary image, the model then tries to recreate a video with proceeding frame after the input frame.**

Keywords—homeostatic plasticity controller, formatting, division into frames, prediction, training & testing

I. INTRODUCTION

The HTM (Hierarchical Temporal Memory) is based on "Thousand Brains Theory" which explains how an object behaviors and high-level concepts gets tightly replicated across a cortical column but not only on the top layer and gets distributed throughout the neocortex. Here spatial pooler involves different computational principles of the cortex (Hawkins, 2017). It depends on competitive Hebbian learning, homeostatic excitability control, topology of connections in sensory cortices and structural plasticity. The HTM Spatial pooler is developed in such a way to achieve a set of computational properties which includes 1. Preserving topology of the input space by mapping similar inputs to similar outputs 2. Continuously adapting to changing statistics of the input stream 3. Forming fixed sparsity representations 4. Being robust to noise and 5. Being fault tolerant that supports computations with SDRs (Sparse Distributed Representations). The output of the SP which is the integral component of HTM can be easily recognized by downstream neurons and contribute to improved performance in the end-to-end HTM system.

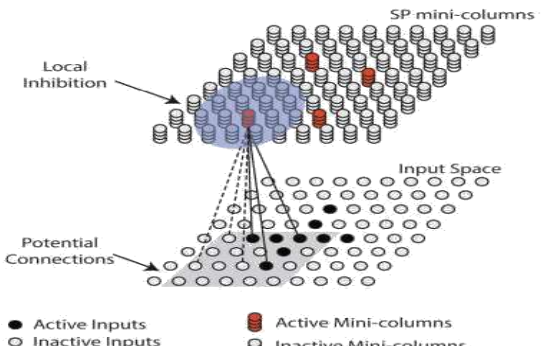

Figure 1: The process of spatial pooling

task of SP is to transforms input patterns into Sparse Distributed Representation in a continuous way in end-to-end HTM system. The temporal sequences of these SDRs is learned by the HTM and do some prediction for the upcoming inputs. A single layer in HTM network is consist of a set of mini-columns which is consist of cells. Here the figure 1. shows that the HTM spatial pooler converts inputs at bottom to SDRs at top. Each SP mini-column (Active mini-columns and inactive Mini-columns) forms synaptic connections to a subset of the input space which is consist of gray square and potential connections. A local inhibition technique gives confirmation that within the local inhibition radius (shaded blue circle) a small fraction of the SP mini-columns that receive most of the inputs are active. According to the Hebbian rule Synaptic permanences are adjusted like this for each active SP mini-column, active inputs (black lines) are reinforced and inactive inputs (dashed lines)are punished (Boudreau, 2018).

At the time of building intelligent systems to mimic human intelligence and cognition, we must pay serious attention to sequences, including sequence learning as sequential behavior is essential to intelligence. In the task of time series prediction, video analysis and musical information retrieval, a model must learn from inputs that are sequences. Another important concern in sequence learning is hierarchical structuring of sequences. Many real-world problems that have sequences are involved with clear hierarchical structures like a sequence is made up of subsequences and they in turn are made up of sub subsequences and so on. By removing the difficulty to identify automatically these subsequences and deal with them accordingly which is related to temporal dependences, learning hierarchical structures help to reduce or eliminate temporal dependencies. It helps to compress the description or sequences.

Sequence learning is not a easy task. Sequence learning are needed powerful algorithms. Sequence learning which indicates either generation, prediction or recognition is usually based on the models of legitimate sequences which can be developed through training with exemplars (Lipton & Berkowitz, 2015). Hierarchical Temporal Memory proposed new computational learning models, Cortical Learning Algorithms (CLA), that is inspired from the neocortex which offer a better understanding of how our brains function. CLA mimics the procedure of human brain how to achieve pattern recognition and make intelligent predictions. The CLA processes the streams of information, classify them, learning to identify the differences and using time-based patterns to make predictions as like as performed by the neocortex in humans. But the place of time is significant in case of learning, inference and prediction. The temporal sequence is achieved from HTM algorithm from the stream of input data.

Here Afterwards the result of the learning is tested by giving the trained model an arbitrary image, the model then attempts to recreate a video with proceeding frame after the input frame.

This type of works forecasting that Machine Learning (ML) or statistical modelling emphasis here is to enable the reader to understand on some of ML or statistical techniques actively used in past and till the present moment.

II. METHODS

A. *Creating Video Data Files*

There are different ways to create training videos of object recognition but we chose to create our object videos using python OpenCV library. As we worked on the previous "**neocortexapi-videolearning**" project we had video data set for recognizing circle, triangle and rectangle. With the help of previous python codes we created training video-set for a line moving around the 120x120 frame with a frame rate of 24 frames per second and the thickness of the object is 8. Videos can be found "SmallTrainingSet" folder.

B. *Reading Video*

To train a video to a machine learning program it has to be divided it into picture frames. As like brain, frames are the point of reference to the recognition program. As we are predicting future frames of an object or in plain language next move of the object according to it's behavior we need to give as small distinctive data as possible to the program for reducing computational time. By VideoSet class from the supporting library of the project VideoLibrary we extract frames of each video.

As the previous videos of this training data set had the same frame rate as our line data the video configuration is not changed. We implemented a videoConfig.json file for the ease of configuration changes to the videos implemented in the future for training. The training video rate is reduced to half of the original video frame rate(see Table **1**)for making more frames and for computational ease of data to be introduced in the system the frame size is also reduced to 18x18 pixels.

Table 1: Video Configuration in videoConfig.json file. Here frame rate is half of the original frame rate.

```
frameWidth": 18,
frameHeight": 18,
frameRate: 12,
ColorMode: "BLACKWHITE"
```

C. *Converting Frames to bitarrays:*

For learning in the spatial pooler and temporal memory each frame has to be binarized. By NFrame class we binarized each frame into binary array using BitmapToBinaryArray method. (see Table **2**) We used Black&White format to binarize each frame as our training videos are created based on Black and White color mode for ease of computational time required for processors. Pure or RGB color mode requires great amount of computational time as per our tests. When the predicted images are recreated IntArrayToBitmap method is used.

Table 2: Frame to bit array calculation in code. Luminance and pixels are binarized with this logic.

```
private int[] BitmapToBinaryArray(Bitmap image)
{
for (int heightCount = 0; heightCount < img.Height; heightCount++)
   {
    for (int widthCount = 0; widthCount < img.Width; widthCount++)
       {
        switch (colorMode)
           { imageBinary.Add((luminance > 255 / 2) ? 0 : 1);
             imageBinary.AddRange(new List<int>() { (pixel.R > 255 /
2) ? 1 : 0, (pixel.G > 255 / 2) ? 1 : 0, (pixel.B > 255 / 2) ? 1 : 0 });
             imageBinary.AddRange(ColorChannelToBinList(pixel.R));
             imageBinary.AddRange(ColorChannelToBinList(pixel.G));
             imageBinary.AddRange(ColorChannelToBinList(pixel.B));
           }
       }
   }
}
```

D. *Learning Stage*

Now the binarized frames are sent to Spatial Pooler which operates on mini-columns to sensory inputs (for this case the movement of the object, it's thickness, size, direction etc.) and learn spatial patterns by encoding the pattern into Sparse Distributed Representation (SDR)

0	…	1	1	1	0	…	1	1	1
1	1	0	…	1	1	1	1	1	1
0	…	1	1	1	1	1	1	0	…
1	1	1	1	1	0	…	1	1	1
1	0	…							

Figure 2: SDR representation of Circle frame no- 0. (Here in the places of ... holds consecutive 0s)

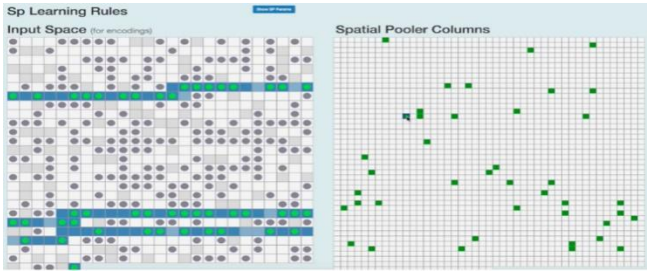

Figure 3: Representation of SDR in spatial pooler columns

The created SDR which is the encoded spatial pattern of that object is used as the input to the Temporal Memory which learns about the patter when the spatial pooler is instable mode and removes the pattern when it is in unstable mode. SP oscillates between stable and unstable mode and the TM also learns and forgets about the pattern. But too much oscillation can cause permanent disruption to the program

hence causing higher computational resources. To reduce this scenario we used homeostatic plasticity controller which influences excitation and inhibition balance of neurons. The functional stability of neural columns is achieved by SP and TM setting cells in active or predictive state. SP provides Global and Local inhibition which controls the number of cells must be activated in the currently processing area. To keep the stability of the Spatial Pooler and learning of TM a set of common parameters were selected while instantiating HTM (see Table **3**) and kept in the htmConfig.json file. Some of the configurations are manipulated while running the program in ModifyHtmFromCode method in the Main Program class.

Table 3: HTM (Hierarchical Temporal Memory) Configuration in htmConfig.json file to initiate HtmCofig class.

Parameters	Value
CellsPerColumn	30
GlobalInhibition	true
NumActiveColumnsPerInhArea	0.02*ColumnDimension
PotentialRadius	0.15* InputDimension
MaxBoost	30.0
MaxSynapsesPerSegment	0.02*ColumnDimension
Random	42
MinPctOverlapDutyCycles	0.75
DutyCyclePeriod	100
StimulusThreshold	0.05*ColumnDimension
UpdatePeriod	50
PermanenceIncrement	0.1
PermanenceDecrement	0.01

Now the boosting in spatial pooler makes sure that all columns are uniformly used across all seen patterns. As the mechanism remains active throughout the process the boosting of columns which already build learned SDRs is possible (Dobrick, 2021). Deactivation of boosting in homeostatic plasticity in the cortical layer can also be applied to SP. But the actual understanding to this is yet to be revealed. Till now in HTM this technique consists of boosting and inhibition algorithms which works on the minimum column level and not on the cell level in the minimum column. Because SP operates on the population of neural cells in minimum column rather than the individual cells(see Figure **2**). Therefore, the Spatial Pooler with the New-born Stage is used with the aim to send input pattern of SDR in each iteration to the homeostatic plasticity controller telling the program that SP has reached instable stage and program will disable the boosting. As the SP has entered to a stable state it will leave the new-born cycle and continue operating as usual without boosting which will help in reducing computational time.

For differentiating multi sequence learning and sequence learning we instantiated HtmClassifier with two different approaches. In the sequence learning method defined in VideoLearning class as TrainWithFrameKey, we put the frame key as HtmClassifier key while calling for the learn method. On the other hand for sequential learning we used series of frame as the HtmClassifier key while calling for the learn method (see Table **4**). By the definition sequence learning should take more computational time while learning as it learns by each frame. But the multi sequence learning should take less time as it takes a bunch of frames while learning.

Table 4: Code for training with frame key in sequential learning and with series of frame keys in multisequence learning.

```
public static void TrainWithFrameKey(VideoConfig videoConfig = null, HtmConfig htmCfg = null)
{
   HtmClassifier<string, ComputeCycle> cls = new();
   HomeostaticPlasticityController hpa = new(mem, maxNumOfElementsInSequence * 150 * 3, (isStable, numPatterns, actColAvg, seenInputs) =>{}, numOfCyclesToWaitOnChange: 50)
   SpatialPoolerMT sp = new(hpa);
   for (int i = 0; i < maxCycles; i++)
   {
      foreach (var currentFrame in nv.nFrames)
         {
            cls.Learn(currentFrame.FrameKey, actCells.ToArray());
            cls.Learn(key, actCells.ToArray()); //For TrainWithFrameKeys
         }
   }
}
```

E. Predicting Frames from an input

After the learning we counted the accuracy of each learned video by calling a method PredictImageInput of VideoLearning class which takes an image and recreates the consecutive frames after that image. This is done in two stages. First from image directories given in the videoConfig.json file (see Table **5**) and then taking images as directory path from users. The directory holding testing videos without input from user contains frames created by the program as it is required to test from the given input also.

As all the data is binarized, these images also needs to be binarized with NFrame class's BitmapToBinaryArray method. While making the predicted future frames IntArrayToBitmap method of the same class is used and then combining all of these frames are done in NVideo class's CreateVideoFromFrames method and saves those in a directory called convertedVideoDir. As the program was built on old version Emgu.cv framework, this method used to have -1 called while initiating VideoWriter object for the manual selection of coding-decoding format of the video which is now obsolete. Now user can select the video format while calling CreateVideoFromFrames method as we introduced fourcc for format selection(see Table **6**)but the default is mp4 format.

We also have calculated the accuracy on the training dataset as well as the testing data set. The accuracy is calculated using equation:

$$accuracy = \frac{Matches\ Found}{Number\ of\ Frames} \times 100 \quad (1)$$

The accuracy reaches to saturation and after getting 10 similar accuracy the program moves to next cycle to reduce computational time(see Table **7**). If the accuracy is more than 80% then it is recorded with the predicted video.

Table 5: Test files containing frames to be tested in videoConfig.json file.

```
"TestFiles":[
"TestImageSet\\Converted\\Circle\\circle\\Circle_circle_3.png",
"TestImageSet\\Converted\\Circle\\circle\\Circle_circle_2.png",
"TestImageSet\\Converted\\Line\\line\\Line_line_11.png",
"TestImageSet\\Converted\\Line\\line\\Line_line_22.png",
"TestImageSet\\Converted\\Rectangle\\rectangle\
\Rectangle_rectangle_28.png",
"TestImageSet\\Converted\\Rectangle\\rectangle\
\Rectangle_rectangle_18.png",
"TestImageSet\\Converted\\Triangle\\triangle\\Triangle_triangle_23.png",
"TestImageSet\\Converted\\Triangle\\triangle\\Triangle_triangle_0.png"
]
```

Table 6: Recreating video on predicted frames by using CreateVideoFromFrames method of NVideo class.

```
public static void CreateVideoFromFrames(List<NFrame> bitmapList,
string videoOutputPath, int frameRate, Size dimension, bool isColor,
char[] codec = null )

{

int fourcc = VideoWriter.Fourcc(codec[0], codec[1], codec[2], codec[3]);

using (VideoWriter videoWriter = new("videoOutputPath..mp4", fourcc,
(int)frameRate, dimension, isColor))

}
```

Table 7: Accuracy check code for both Sequential and Multisequence learning in VideoLearning class.

```
// Accuracy Check
double accuracy;
accuracy = (double)matches / ((double)nv.nFrames.Count - 1.0) *
100.0;
if (accuracy == lastCycleAccuracy)
{
   saturatedAccuracyCount += 1;
   if (saturatedAccuracyCount >= 10 && lastCycleAccuracy >= 80)
   {
   }
}
```

III. Result

Before we directly reach to accuracy result first have to understand about the changes we made algorithmically. In the previous version of the code all the prediction were made in a do while loop where a new user won't understand when to put image input after an input iteration and the program used to be at a stand still position without any instruction. As a migration of this code we made the program user friendly starting with all the instructions required to start the program and where to change if required which is given at the starting of the program and where instructions are required to run further(see Table **8**).

There was an unnecessary HelperFunction class previously which required extra memory, we integrated it into the main VideoLearning class. Also created methods to reduce redundancy and follow dry(Don't repeat yourself) technique like MakeDirectoryIfRequired, GetVideoSetPaths etc. methods.

Most of the previously created functions and methods had missing summary issues and we described those in summary so that a programmer in future using this library can easily instantiate those methods and functions by reading.

Table 8: Initial Instruction running the code with full details of where to change.

```
WriteLineColor($"Hello NeoCortexApi! Conducting experiment
{nameof(VideoLearning)} CodeBreakers" + "\n" +
"This program can take initial information of the training video from
VideoConfig.json" + "\n" +
"If you are training with a new set of videos please place the videos in
the folder name SmallTrainingSet" + "\n" +
"Moreover you also need to give video metadata information in the
VideoConfig.json file" + "\n" +
"To change HTMClassifier configuration use htmConfig.json");


WriteLineColor("Drag an image as input to recall the learned Video or
type (Write Q to quit): ");
```

A. *Accuracy Check*

We had total four sets of video data. The videos can be found in SmallTrainingSet folder. Each of the type is separated into 4 different folders named as Circle, Line, Rectangle and Triangle. As the calculated accuracy is also logged using the function UpdateAccuracy of VideoLearning class, it is easy to find out where the accuracy got saturated value and how much time it required to reach the saturated value. For accuracy result collection we used 1000 cycles maximum to learn and predict frames.

Table 9: the accuracy result table for TrainWithFrameKey, Sequential Learning.

Data Type	Highest Accuracy	Saturated Accuracy
Circle 	102.857% Stability reached at 185th newborn cycle	102.85714285% Saturation level fixed at 117th cycle
Line	100% Stability reached at 185th newborn cycle	95.74468085% Saturation level fixed at 447th cycle
 Rectangle	100% Stability reached at 185th newborn cycle	100% Saturation level fixed at 144th cycle
Triangle 	100% Stability reached at 185th newborn cycle	100% Saturation level fixed at 127th cycle

Table 10: The accuracy result table for TrainWithFrameKeys, Multisequence Learning

Data Type	Highest Accuracy	Saturated Accuracy
Circle 	102.857% Stability reached at 177th newborn cycle	102.8571428% Saturation level fixed at 69th cycle

Line	97.87% Stability reached at 177th newborn cycle	97.8723404% Saturation level fixed at 217th cycle
Rectangle	100% Stability reached at 177th newborn cycle	97.1428571% Saturation level fixed at 77th cycle
Triangle	100% Stability reached at 177th newborn cycle	100% Saturation level fixed at 17th cycle

From these table we can easily say that multisequence learning where couple of frames are pushed together in the learning stage reaches to saturation in less iterations than the sequential learning. It is because the multisequence learning reduces frames iteration while learning and SDR patterns can be easily captured. In the sequential learning prediction is done with only one possible outcome. But in case of multisequence learning prediction is done for 3 possible outputs(see Table **11**). For this reason highest accuracy in case of sequential learning is higher than the multisequence learning. But considering overall situation regarding computational time, computational resources' requirement, accuracy reaching time and required cycles we can say that multisequence learning for video learning projects are the best solution in between sequential learning and multi sequence learning.

Table 11: Code for prediction for each frame in VideoLearning class.

```
foreach (VideoSet vd in videoData)
{
   foreach (NVideo nv in vd.nVideoList)
   {
      for (int i = 0; i < maxCycles; i++)
      {
         foreach (var currentFrame in nv.nFrames)
         {
            if (lyrOut.PredictiveCells.Count > 0)
            {
               var predictedInputValues =
cls.GetPredictedInputValues(lyrOut.PredictiveCells.ToArray(), 1);
// Ii is for the method TrainWithFrameKey which is sequential Learning
               Var predictedInputValues =
cls.GetPredictedInputValues(lyrOut.PredictiveCells.ToArray(), 3);
// Ii is for the method TrainWithFrameKeys which is multisequence
Learning

            }
         }
      }
   }
}
```

IV. Discussion

As this is the migration of the old Video Learning project we have changed a lot in it's learning and prediction algorithm. We have also devided the VideoLearning library into three useable functions TrainWithFrameKey, PredictImageInput and TrainWithFrameKeys by which one can write any new Video Learning project easily. For the ease of understanding we added summary to every functions and methods. But still the accuracy of the project incase of separate image input from user is not up to the mark. As the last learned video set for this project is Triangle if we put a frame from the line it recognizes the pattern from triangle frames. This is also like our brain, when we read something most of the cases we can remember the last thing we have learned more accurately than the previously seen things. By the log file we can easily see that the accuracy oscillated in the prediction stage. This is also because HTM forgot some of the frames in the learning stage. We found the optimal forgetting and learning ratio as 1/10. Finding a better ratio requires more research on this project. Also different video sets requires different configuration of HtmConfig class according to video configuration introduced in the videoConfig.json file.

There are still lot of possible improvements to this video learning project. Currently we took the video codec (coding-decoding) format mp4 but new and improved codec is introduced in the libraries and many more are coming. Finding a suitable codec like wmv or MPEG-4 AVC can improve the video making after the prediction more accurately.

Most of the predicted video has unwanted edges around the object. This happened because we have used System library's Drawing class. It also has a draw back of not being OS independent meaning that this will only run on windows but not in linux. There is a new library called SkiaSharp which is based on Google's Skia Graphics Library and it is OS independent with more accuracy while building the frames from encoded bits. This can help in reducing the edges and building this program on cloud based or OS independent environments.

V. References

- Boudreau, L. G. (2018, 5). *Rochester Institute of TechnologyRochester Institute.* Retrieved from RIT Scholar Works: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=10897&context=theses
- Dobrick, D. (2021). *Improved HTM Spatial Pooler with.* Retrieved from University of Plymouth: https://pearl.plymouth.ac.uk/bitstream/handle/10026.1/17130/Improved%20HTM%20spatial%20pooler%20with%20homeostatic%20plasticity%20control.pdf?sequence=1&isAllowed=y
- Hawkins, S. &. (2017, 10 17). *A Theory of How Columns in the Neocortex Enable Learning the Structure of the World.* Retrieved from https://frontiersin.org/articles/10.3389/fncir.2017.0008
- Lipton, Z. C., & Berkowitz, J. (2015). *A Critical Review of Recurrent Neural Networks*, 2-4.