**Implement HTM Persistance**

Kizito Onyema  
kizito.onyema@stud.fra-uas.de

Nabeela Maham  
nabeela.maham@stud.fra-uas.de

*Abstract*— The aim of the project is to is implement and design UnitTests that could aid the NeocortexApi to Serialize and Deserialize its HTM Objects so that it can recall from where it stops. The Serializer records the instances of some of the HTM module to the stream and the Deserializer helps in creating the instance from the stream.

This work deals with the implementation, design and conducting of UnitTests in two ways; by means of Serialization where by the HTM keeps record of the instances of the HTM modules and continues again from it’s last instance by Deserialization. As this is a new feature in the NeoCortexApi, the current implemented and conducted tests are still limited. As the NeoCortexApi is the mirror of a Human Brain, this work describes an aspect of the real life where a Human’s Brain coordinates itself and reminds the body of it last work continues his/her Work where it stoped; this emphasizes the NeoCortexApi’s cognitive ability, remembrance, continuity and efficiency.

Keywords— - serialization, deserilization, persistence

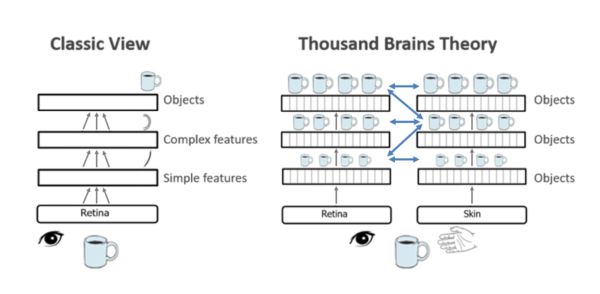
# **Intro (*Heading 1*)**

Hierarchical temporal memory (HTM) is a biologically constrained machine intelligence technology developed by Numenta. Originally described in the 2004 book. On Intelligence by Jeff Hawkins with Sandra Blakeslee, HTM is the distributed building and primarily used today for anomaly detection in streaming data. The technology is based on neuroscience and the physiology and interaction of pyramidal neurons in the neocortex of the mammalian (in particular, human) brain.

The HTM algorithm is based on the well understood principles and core building blocks of the Thousand Brains Theory. In particular, it focuses on three main properties: sequence learning, continual learning and sparse distributed representations.

The Thousand Brains Theory is the core model-based, sensory-motor framework of intelligence putting together the neuroscience research developed over the almost two decades of research at Numenta and the Redwood Neuroscience Institute (founded in 2002 by Jeff Hawkins).

It provides a unique interpretation of the high-level computation thought to happen throughout the neocortex and giving rise to intelligent behaviors.



Since the HTM is based on the ideology of the neocortex in the Human Brain which has the ability of remembrance and continuity, such a function can also be created implemented as Persistence in the NeoCortexApi which could be achieved through serialization and deserialization; this means that the instances created by the HTM Modules a recorded at a point, and also willing to continue at any point in time with these points. We apply the serialization to save the data in the form of text file. We can use that data next time when you continue the experiment again in deserialization. The goal here is to improve the efficiency and remembrance functionality of the NeoCortexApi.

For the serialization, we have used the class HTMSerializer2.cs. There, we have applied different functions. It has serialized the property of integer, double, string, long, boolean and fetch the values for the deserialization. It can serialize the non-primitive data types such as cell, dictionary distal dendrite, synapse and list. Deserialize the value when we need to continue the experiment again.

# **Methodology**

SERIALIZATION

Serialization is the process of converting an object into a stream of bytes to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

## How serialization works

This illustration shows the overall process of serialization:



The object is serialized to a stream that carries the data. The stream may also have information about the object's type, such as its version, culture, and assembly name. From that stream, the object can be stored in a database, a file, or memory.

### Uses for serialization

Serialization allows the developer to save the state of an object and re-create it as needed, providing storage of objects as well as data exchange. Through serialization, a developer can perform actions such as:

* Sending the object to a remote application by using a web service
* Passing an object from one domain to another
* Passing an object through a firewall as a JSON or XML string
* Maintaining security or user-specific information across applications

*Serialization in HTM Persistance:*

#region Serialization

public void Serialize(StreamWriter writer)

{

HtmSerializer2 ser = new HtmSerializer2();

ser.SerializeBegin(nameof(SegmentActivity), writer);

ser.SerializeValue(this.ActiveSynapses, writer);

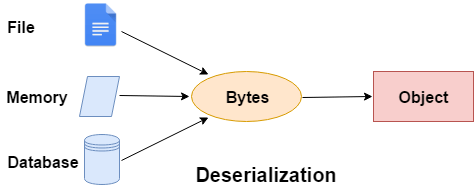
ser.SerializeValue(this.PotentialSynapses, writer);

ser.SerializeEnd(nameof(SegmentActivity), writer);

}

# Deserialization

Deserialization is the reverse process of serialization. It means you can read the object from byte stream. Here, we are going to use the method to deserialize the stream



### Deserialization Example

*Derialization in HTM Persiatnce*

public static SegmentActivity Deserialize(StreamReader sr)

{

SegmentActivity segment = new SegmentActivity();

HtmSerializer2 ser = new HtmSerializer2();

while (sr.Peek() >= 0)

{

string data = sr.ReadLine();

if (data == ser.LineDelimiter || data == ser.ReadBegin(nameof(SegmentActivity)) || data == ser.ReadEnd(nameof(SegmentActivity)))

{ }

else

{

string[] str = data.Split(HtmSerializer2.ParameterDelimiter);

for (int i = 0; i < str.Length; i++)

{

switch (i)

{

case 0:

{

segment.ActiveSynapses = ser.ReadDictionaryIIValue(str[i]);

break;

}

case 1:

{

segment.PotentialSynapses = ser.ReadDictionaryIIValue(str[i]);

break;

}

default:

{ break; }

}

}

}

}

return segment;

}

The project in the folder HTMPersistanceUnitTest



public Synapse CreateSynapse(int index, int inputIndex)

*Classes in this Project:*

I have Tested three classes in this project. These classes are listed below.

*1. Segment Activity:*

SegmentActivity.cs stores the calculus of a temporal cycle. It contains the index of segments with number of synapses with permanence higher than threshold. A Dictionary, which holds the number of potential synapses of every segment.

   public override void Serialize(StreamWriter writer)

  public static ProximalDendrite Deserialize(StreamReader sr)

Potential synspses are all established synapses between receptor cell and the segment's cell. Receprot cell was active cell in the previous cycle.

link to [SegmentActivity](https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/SegmentActivity.cs)

Dictionary[segment index, number of active synapses].

Dictionary [segment index, number of potential synapses]

Serialization and deserialization have been applied on SegmentActivity.

*2. Proximal Dendrite:*

It defines the proximal dendrite segment. Note the segment is used during SP compute operation.

TM does not use this segment.

It uses the pool of synapses in the receptive field.

Listing 1 Constructor from [ProximalDendrite](https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/ProximalDentrite.cs) class

public ProximalDendrite(int colIndx, double synapsePermConnected, int numInputs) : base(colIndx, synapsePermConnected, numInputs)

It creates and returns a newly created synapse with the specified source cell, permanence, and index. This method is only called for Proximal Synapses. For ProximalDendrites, there are many synapses within a pool, and in that case, the index specifies the synapse's sequence order within the pool object, and may be referenced by that index. It returns the instance of the new synapse.

Indicies of Array of connected inputs defines RF(Potential Pool). It clear all the synapses from the segment. Sets the permanences for each linked Synapse specified by the indexes passed in which identify the input vector indexes associated with the permanences passed in are understood to be in "sparse" format and therefore require the int array identify their corresponding indexes. This is the "sparse" version of this method. Returns an array of synapse indexes as a dense binary array. Returns an array of indexes of input neurons connected to this pool. It returns the indexes of connected input neurons.

*3. Distal Dendrite:*

Implements a distal dendritic segment that is used for learning sequences. Segments are owned by Cells and in turn own Cells which are obversely connected to by a "source cell", which is the Cell that will activate a given Synapse owned by this Segment.

link to [DistalDentriteDendrite](https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/NeoCortexEntities/Entities/DistalDendrite.cs)

  public class DistalDendrite : Segment, IComparable<DistalDendrite>, IEquatable<DistalDendrite>

**Cell:** The cell that owns (parent) the segment.  the last iteration in which this segment was active.

The sequence number of the segment. Specifies the order of the segment of the Connections instance.

*ParentCell:*

 The cell, which owns the segment.

*flatIdx:*

The flat index of the segment. If some segments are destroyed (synapses lost permanence)then the new segment will reuse the flat index. In contrast, the ordinal number will increas when new segments are created.

*lastUsedIteration*

*ordinal:*

The ordinal number of the segment. This number is incremented on each new segment.

If some segments are destroyed, this number is still increment.

*synapsePermConnected*

*numInputs*

public DistalDendrite(Cell parentCell, int flatIdx, long lastUsedIteration, int ordinal, double synapsePermConnected, int numInputs) : base(flatIdx, synapsePermConnected, numInputs)

It compares this segment with the given one and Compares by index as well.

It  Serialize method for DistalDendrite

internal void SerializeT(StreamWriter writer)

Then apply deserialization.

public static DistalDendrite Deserialize(StreamReader sr)

*Unit Test Classes for the project.*

The following code is applid for testing the classes(**SegmentActivity, DistelDendrite, pxorimalDendrite**)

Here is the link to the [unit Test class.](https://github.com/nabeelamaham/neocortexapi/blob/Nabeela-HTMPersistance/source/HTMPersistanceUnitTests/SerializeSegmentActivityTest.cs)

     public void TestSegmentActivitySErialization()

        [TestClass]

        public class DistalDendriteSerializationTest

        [TestClass]

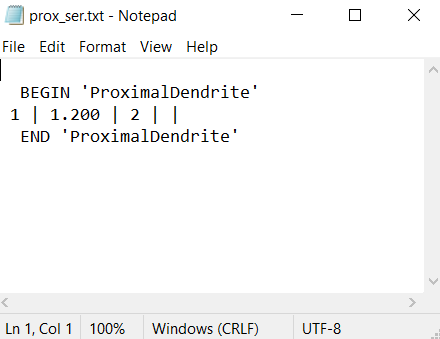
        public class ProximalDendriteSerializationTest

*Serialization logic*

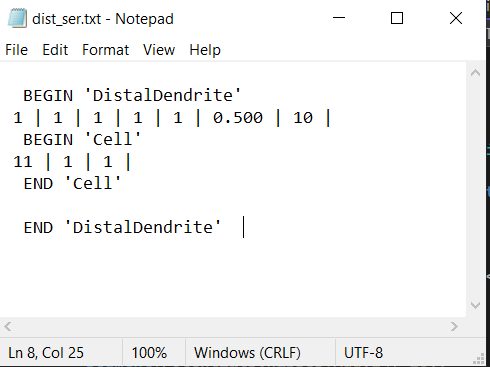
It is used for serialization and deserialization of primitive types. Such as Integer, Boolean, String, Array Int[], Double, Long. It work for non primitive type such as Synapses and cells.

 Serializes the begin and end marker of the type.

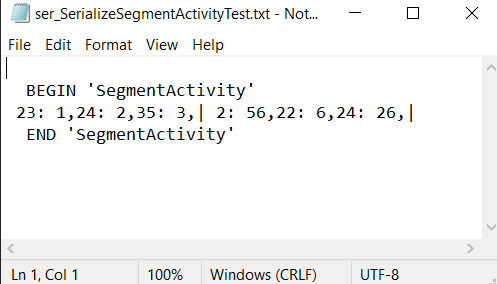
*ProximalDendrite.txt:*

**

*DistalDendrite.txt:*

**

*SegmentActivity.txt:*

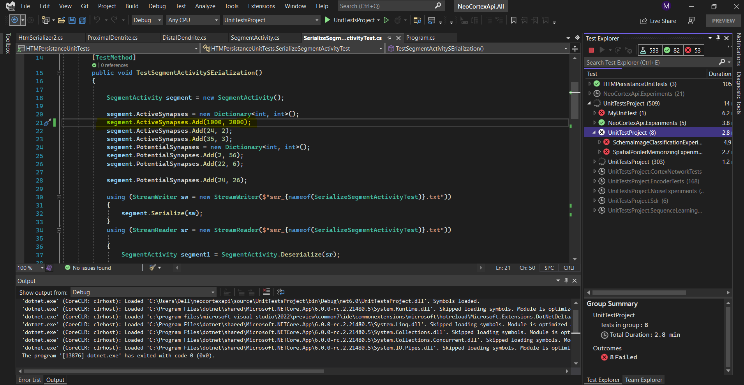


 Htm persistance serialize the property of type Int. Read the property of type Int and return that integer. Deserializes from text file to DistalDendrite and return DistalDendrite. Serialize the property of type Double, String, Long and Bool. Read the property of type Double, String, Long, Bool and return these values. Serialize the array of type Double. Read the array of type Double and return double. Serialize the array of type Int. Read the array of type Int returns Int[]. Serialize and Deserialize the array of cells. Deserializes from text file to Cell and return cells. Serialize the dictionary with key:string and value:int. Read the dictionary with key:string and value:int and return Dictionary<String, int>. Serialize the List of DistalDendrite, Synapses and Integer and Dictionary<Segment, List<Synapses>>. Read the List of DistalDendrite and returns distal dendrite. Serialize the dictionary and Concurrentdictionary with key:int and value:Synapse and DistalDendrite. Read the dictionary with key:int and value:Synapse return Dictionary<int, Synapse>

# **Results**

We have applied different values to know the accuracy in the project or in the unit test class.

I used the different value on the ActiveSynapse that is highlighted with yellow and it is showing the result in the screenshot below. Unit test class is showing an error using this value.



The following classes are not implemented in this project.

* AbstractMatric,
* AbstractFlatMatrix,
* SpatialPooler
* TemporalMemory
* Column,
* Synapse,
* Topology
* Pool
* Connections

# **Discussion**

In the conclusion, I could say that the implementation of HTM Persistence can save a lot of time. In addition, WE have implement it on visual studio using C sharp but we can work on Yaml, JSON and XML in future and perform serialization and deserialization and get better results as well

In this project, we have worked on text file. We have applied serialization and deserialization on the txt file. In the futurn when we try to enhance our project we could use JSON serialization. It is a class of Javascript which can share the data across the web. We can use different systems for performing serialization and deserialization.

WE can also apply binary and XML serialization Binary serialization uses binary encoding to produce compact serialization for uses such as storage or socket-based network streams. In binary serialization, all members, even members that are read-only, are serialized, and performance is enhanced.

##### **References**

1. <https://www.guru99.com/c-sharp-serialization.html>
2. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/>
3. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/serialization/walkthrough-persisting-an-object-in-visual-studio>