Implement Temporal Memory Learning sample with Serialization

Information Technology Course

Module Software Engineering

by Damir Dobric/ Andreas Pech

Your Name

[hung.buitranhai@stud.fra-uas.de](mailto:hung.buitranhai@stud.fra-uas.de)

# Abstract­­

This paper gives an example how Serialization with Temporal Memory can be used in “MultiSequence Learning” project. In this example, the serialization method is implemented to save the trained model of “MSL” and keep it locally on computer. The de-serialization method is also implemented to later load the last trained model which will be the input for the following training. The project aims to make “Predictor” object, which is the output of the “MSL” be serializable. The paper gives a clear demonstration how we could do it easily with the “HtmClassifier”, a class used for serialization which is already implemented in the “NeoCortexApi” solution. The example made in this project will show how the methods for serialization and de-serialization are used to save and load the trained model, how the last trained model will be used for next training.

# Table of Abbreviation

MSL MultiSequence Learning

SP Spatial Pooler

TM Temporal Memory

# Introduction

# Methods

In order to implement serialization with TM for MSL project, we need at first make the “Predictor” which is the output of MSL become serializable. Hence, the methods used for serialization and de-serialization the Predictor need to be implemented. Since it is difficult to de-serialize “Predictor” from one text file, we are not serializing the “Predictor” to a single file. The approach in this project is to serialize every objects within the “Predictor” to different files, so that we can later easily de-serialize them and reconstruct the “Predictor” from the objects. The advantage of this approach is that we can know easily whether the objects are de-serialized properly with the debuger before the “Predictor” are reconstructed and returned.

The Predictor class inherits from the ISerializable:

* public class Predictor : Iserializable
* public interface ISerializable

{

void Serialize(object obj, string name, StreamWriter sw);

static object Deserialize<T>(StreamReader sr, string name) => throw new NotImplementedException();

}

Therefore, the Serialize() and Deserialize() methods need to be implemented in Predictor class:

* Serialize() method which is implemented in the Predictor class [1].
* Deserialize() method which is implemented in the Predictor class [2].

## Serialization

First, we would like to clarify the approach for serialization.

From [1] we can see that, the serialize() method in the Predictor class are used to serialize objects included in the Predictor(Connections, CortexLayer, and HtmClassifier). That means we have to implement first the serialize() methods in Connections class, CortexLayer class and also HtmClassifier class. Those serialize() methods will then be called in the serialize() method implemented in the Predictor class, so that we can serialize all of the Predictor’s components.

* Serialize() method which is implemented in the Connection class [3].
* Serialize() method which is implemented in the CortexLayer class [4].
* Serialize() method which is implemented in the HtmClassifier class [5].

From [4] we can see that, the CortexLayer consists of three layers which are the Scalar Encoder, the Spatial Pooler, and the Temperal Memmory. Hence, we also need serialize() methods implemented in those classes.

* Serialize() method which is implemented in the EncoderBase class [6].
* Serialize() method which is implemented in the Spatial Pooler class [7].
* Serialize() method which is implemented in the Temporal Memory class [8].

The objects belong to the Predictor are finally serialized to different text files as in Figure 1.

Table

Description automatically generated

Figure 1: The serialized objects are saved locally

## Deserialization

After we are able to serialize the objects in the Predictor, we have to then implement the deserialize() method that can retrieve the objects back from the files and return the last Predictor.

# Results

# Conclusion

# References

1. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/Predictor.cs#L76>
2. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/Predictor.cs#L121>
3. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/76c4979d9ba698231e0e38ea3610a67c851d4f74/source/NeoCortexEntities/Entities/Connections.cs#L1596>
4. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/Network/CortexLayer.cs#L173>
5. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/Classifiers/HtmClassifier.cs#L749>
6. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/Encoders/EncoderBase.cs#L320>
7. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/SpatialPooler.cs#L1438>
8. <https://github.com/Hungbth2000/tml_serialization_mltseq/blob/5697e5954ce8f589d07b43ffcfbc92d16304f380/source/NeoCortexApi/TemporalMemory.cs#L897>