///<summary>

/// Test adapt segment from syapse to centre when synapse is already at the center

/// <Summary>

[TestClass]

public class TemporalMemoryTests

{

[TestMethod]

public void TestAdaptSegmentToCentre\_SynapseAlreadyAtCentre()

TestTemporalMemoryComputeReturnsWinnerCells\

TestIncreasePermanenceOfActiveSynapses()

TestGetLeastUsedCell(

TestColumnDimensions

{

//Arrange

TemporalMemory tm = new TemporalMemory();

Connections cn = new Connections();

Parameters p = Parameters.getAllDefaultParameters();

p.apply(cn);

tm.Init(cn);

DistalDendrite dd = cn.CreateDistalSegment(cn.GetCell(0));

Synapse s1 = cn.CreateSynapse(dd, cn.GetCell(23), 0.6); // central

//Act

TemporalMemory.AdaptSegment(cn, dd, cn.GetCells(new int[] { 23 }), cn.HtmConfig.PermanenceIncrement, cn.HtmConfig.PermanenceDecrement);

//Assert

Assert.AreEqual(0.7, s1.Permanence, 0.1);

}

}

The code snippet provided is a unit test for a method called "AdaptSegmentToCentre" in a Temporal Memory class. This method is used to adjust the permanence value of synapses in a distal dendrite segment to enhance the prediction accuracy of the temporal memory algorithm. The unit test aims to verify that the method works correctly when a synapse is already located at the center of the segment. The unit test method is named "TestAdaptSegmentToCentre\_SynapseAlreadyAtCentre". It is decorated with the "TestMethod" attribute to indicate that it is a unit test. The method first initializes a temporal memory instance and sets up the necessary parameters for the algorithm. Then, it creates a distal dendrite segment and adds a synapse with a permanence value of 0.6 to the center of the segment. After setting up the initial state, the unit test calls the "AdaptSegment" method with the created segment, the cell located at the center, and the permanence increment and decrement values from the configuration. The purpose of this method is to adjust the permanence values of the synapses in the segment to strengthen the connections between the cells that are active in the current input pattern.Finally, the unit test checks if the permanence value of the synapse created in the setup stage is increased by 0.1, which is the permanence increment value in the current configuration. The expected value is 0.7, and the test verifies that the actual value of the synapse's permanence is within a 0.1 tolerance of the expected value using the "Assert.AreEqual" method. The unit test verifies that the "AdaptSegment" method works correctly when a synapse is already located at the center of the distal dendrite segment. It ensures that the permanence value of the synapse is increased by the configured amount, which indicates that the method correctly identifies and strengthens the existing connections between the cells. The unit test for the "AdaptSegmentToCentre" method with a synapse already at the center of the segment verifies that the method works correctly and produces the expected results. This is an important test case, as it ensures that the algorithm can accurately adapt the segment to improve prediction accuracy even when there is an existing connection at the center of the segment.

[TestMethod]

public void TestTemporalMemoryComputeReturnsWinnerCells()

{

TemporalMemory tm = new TemporalMemory();

Connections cn = new Connections();

Parameters p = getDefaultParameters(null, KEY.CELLS\_PER\_COLUMN, 2);

p = getDefaultParameters(p, KEY.MIN\_THRESHOLD, 2);

p.apply(cn);

tm.Init(cn);

int[] activeColumns = { 0, 1, 2, 3 };

ComputeCycle cc = tm.Compute(activeColumns, true) as ComputeCycle;

List<Cell> winnerCells = new List<Cell>(cc.WinnerCells);

Assert.AreEqual(4, winnerCells.Count);

Assert.AreEqual(0, winnerCells[0].Index);

Assert.AreEqual(2, winnerCells[1].Index);

}

------------------------------------------------------------------------------------------

The code snippet provided is a unit test for a method called "Compute" in a Temporal Memory class. This method computes the active and winner cells in a temporal memory based on the input pattern and the current state of the memory. The unit test aims to verify that the "Compute" method returns the correct set of winner cells given a specific input pattern and the current state of the temporal memory. The unit test method is named "TestTemporalMemoryComputeReturnsWinnerCells". It is decorated with the "TestMethod" attribute to indicate that it is a unit test. The method first initializes a temporal memory instance and sets up the necessary parameters for the algorithm. The parameters are set to have two cells per column and a minimum threshold of two. Then, it creates a connections instance based on the parameters and initializes the temporal memory instance with the connections instance. After setting up the initial state, the unit test calls the "Compute" method with an array of active columns representing the input pattern and the "true" value for the learn parameter, which indicates that the algorithm should adjust its permanence values during the computation. The method returns a ComputeCycle object that contains the winner cells and other computed values. Finally, the unit test checks that the number of winner cells is four, which is the number of active columns. It then verifies that the first winner cell has an index of 0, and the second winner cell has an index of 2. These expected values are based on the input pattern and the configured parameters. The unit test verifies that the "Compute" method correctly identifies the winner cells given a specific input pattern and the current state of the temporal memory. It ensures that the algorithm correctly adjusts the permanence values of the synapses and determines the cells that have the strongest connections with the active columns.The unit test for the "Compute" method in the Temporal Memory class verifies that the method returns the correct set of winner cells given a specific input pattern and the current state of the temporal memory. This is an important test case, as it ensures that the algorithm can accurately identify the cells that are most likely to be activated in the next time step based on the current input pattern and the history of the input patterns.