V

Building artificial Neurons in three-dimensional space

# Introduction

We have discussed the possibility of three-dimensional mesh neural networks and the potential for artificial Neurons. In doing so we have touched on some issues we may encounter in larger mesh networks and security issues whilst developing and designing such functionality.

During the second iteration, we will be building Neurons and deploying artificial intelligence within their core to called as a call-back function. The artificial intelligence will look at deploying and making use of MST, logging, reporting and providing return information for analysis.

We will deploy a 32 Neuron setup with various Neuron attributes and make use of a simple dictionary data set with specific output requirements. These Neurons will simply be housed within a mathematical bounded three-dimensional space and have all their respective connectivity’s in place either hard coded or autonomous if we deploy them at random space points within the volume.

In doing so we will also look at the dormant Neurons within the volume and research a best practice to randomise with a little more even distribution in a volume whilst still maintaining ‘organic’ distribution. We will make use of standardised algorithms to assess their functionality and examine efficiency of both deployment at random compared to hard coded and in comparison, to existing results as if the artificial intelligence algorithm were simply a single entity.

With that in mind, we will perform those tasks to examine whether artificially, more Neurons are better than one before proceeding to examine whether all Neurons should have the same artificial intelligence or whether clusters of variable artificial intelligence can interact well.

They will be built in a shared folder on a guest virtual machine operating system which has had the virtual networking card stripped out after the updates and the necessary IDE installed. For now, the development process will be localised on the machine and not run until completed.

The purpose of the second iteration is to refute the Spencer-Harper [2015] statement “It’s not necessary to model the biological complexity of the human brain at a molecular level, just its higher-level rules”. Because sometimes, due to the complexity of the brain (processing several interpretations [senses] of the environment), it is necessary to compile the individual processing lobes (volumes) to replicate and manage tasks in replication to the functionality to opposition to masses of algebraic equations and have them ‘talk’ to each other.

Some more potent advantages of the proposed model is cell based reporting and logging including unique identifiers per cell relating to their location inside the volume and specifying the size and randomness of their locations and connectivity to replicate a closer representation of the organic brain.

# Volume

When setting the volume metrics, we have two options; We can set mathematical bounds such as ten cubed giving us 1000 dots matrices to work with or we can use three dimensional ArrayLists which tend to get a bit cumbersome when trying to find out which matrix location each Neuron has. In addition, each matrix location will be housing another multi-dimensional ArrayList and makes for very difficult decompression and confusion in the programmatic section.

What will be best for the volume is to set perhaps a simple mathematical representation yielding mathematical volumetrics and better precision in neural proximities. Furthermore, scalability will be a lot easier to manage when embedding to the hardware. What we end up with are mathematical boundaries and neural proximities pinpointing programmatic memory locations at their neural epicentre.

public static ArrayList<ArrayList<ArrayList<Object>>> hive(int n) {

int i, j;

int x\_axis = n;

int y\_axis = n;

int z\_axis = n;

ArrayList<ArrayList<ArrayList<Object>>> c = new ArrayList<>(x\_axis);

for (i = 0; i <= x\_axis; i++) {

c.add(new ArrayList<ArrayList<Object>>(z\_axis));

for (j = 0; j < z\_axis; j++) {

c.get(i).add(new ArrayList<Object>(y\_axis));

}

}

return c;}

Method describing a three-dimensional ArrayList composition

The issue we will encounter is programmatically indexing the mesh layers and ensuring the correct depth, though, it may be possible to simply set integer type precision for the volume and float data types for the neural proximities permitting better precision when connectivity is assessed.

When building the volume, we can make use of the base code from the first iteration and proof of concept. Each mesh is a layer index Y with a square surface area of Z multiplied by X and each can be stacked upon the other before the connectivity bonds are called.

private static HashMap<Integer, ArrayList<Object>> beehive = new HashMap<>();

static ArrayList<ArrayList<ArrayList<Object>>> cells = new ArrayList<>();

Creation of the hash map Object and a three-dimensional ArrayList

Doing so, permits the mesh to be generated, layer by layer should speed up the randomisation of Neuron placement on the ZX plane, the connections will inevitably take longer as they are generated, compressed and stored per Neuron and in a file. We could then use a Hash Mapping function from the Java API to map the ArrayList indices to an indexed location thus streamlining access to memory locations in the Mapped Index and have an iterable index of the volume.

*/\*\**

*\* Randomly populate the mapped volume with Neurons*

*\* @param neuron*

*\* @param qty - Quantity of neurons*

*\* @param v - Received autonomously from mapping()*

*\* @throws IOException*

*\*/*

*public static void populate(ArrayList<Object> neuron, int qty, double v) throws IOException {*

*System.err.println("Populating volume with Neurons...");*

*Random rand = new Random();*

*for (int i = 0; i <= qty; i++) { int m = rand.nextInt((int) v); beehive.put(m, Neuron.neuron());};*

*System.err.print("Original mapping hashcode after first generation: " + beehive.hashCode() + "\n");*

*NeuralNet.setBeehive(beehive);*

*}*

*/\*\**

*\* Maps the cubic index length to the ArrayList c and calls the populate()*

*\* autonomously to populate the list and the mapping with the Neurons*

*\**

*\* @param neuron - An Objectified ArrayList comprising the AI and areas / proximities*

*\* @param qty - The quantity of Neurons to deploy randomly in your mapped volume*

*\* @param n - The index to be used to calculate the cubic size of the volume*

*\* @return*

*\* @throws IOException*

*\*/*

*public static HashMap<Integer, ArrayList<Object>> mapping(ArrayList<Object> neuron, int qty, int n) throws IOException {*

*System.err.println("Mapping out a hash map of " + n + " cubed...");*

*int v = (int) Math.pow(n, 3);*

*System.err.println("Creating Volume cubic size: " + v + "...");*

*//Call to create out bee hive ready to map out neurons (bees) into random cells*

*HiveNest.hive(v);*

*//Once we have our cells we can begin to populate, them with our bees....*

*HiveNest.populate(neuron, qty, v);*

*//Ensure we have set our beehive*

*NeuralNet.setBeehive(beehive);*

*//Generate our outputLayer initial weights*

*OutputLayer.outputLayerInitialWeights();*

*//Generate our initial values for the output layer*

*//OutputLayer.output();*

*//Set our hidden Layer Weights to a value other than 0.0 as initialisers*

*HiddenLayerWeights.hidden();*

*//Set the value of our first index in the neuron to the training(), that's the brain of the bee...*

*System.err.println("Adding our training module and coordinates ...");*

*for (Entry<Integer, ArrayList<Object>> populated : beehive.entrySet()) {*

*int x = GetX.getx(populated.getKey().intValue());*

*int z = GetZ.getz(populated.getKey().intValue());*

*int y = GetY.gety(populated.getKey().intValue());*

*populated.getValue().set(8, x);*

*populated.getValue().set(9, z);*

*populated.getValue().set(10, y);}*

*return beehive;}*

Compiling and instantiating a three-dimensional list with its indices at each index according to cubic size

## Volume – Batch and Variance

It would be a good idea to ensure the generation, placements and connections are stored in a file for analysis and regeneration. We must make certain during build and test stages the data is stored because if an optimum is found, we can revert to it for hard coding and we can run a batch optimisation from neurological mean average configuration to mean average result comparison and later check the neurological pattern against the expected result.

As such, we may be able to ascertain if there is a generic configuration which yields the optimal results. There are so many possible configurations of three-dimensional neural mesh, with such variance of artificial intelligences, we would have to run such a practice for the high-performance end of the scale too. More on those topics in further discussion, since we have digressed and should turn focus back to Batch and Variance more specifically populating the Indices, accessing them and activating them.

## Volume – Neurons and Training

With a focus on the type of Neuron we are adding to the volume, for the example herein, we are simply making use of ones and zeroes as the training data and deploy a randomised situation of five ones and zeroes to check if the situation fits in with the training data. The training data is a multi-dimensional ArrayList Object of sets of five integer one’s and zeroes per the original training template from Spencer-Harper [2015].

However, what we have tried to achieve in the modified version and port over to Java, is to make use of the object orientated nature of the language and even make use of objectifying data to alleviate the necessity to reprogram areas of the system to suite certain datatypes, though that may still be a nice requirement in future iterations.

The porting is quite difficult since we are rewriting in a different language and often making use of a different API which may or may not contain libraries from the other languages API. Furthermore, we are making use of ArrayLists which are notorious for their variation in data type storage and so modifications to standardised mathematical algorithms had to be made, two prime examples were the transpose function (which may need modifying further) -

public class Transpose {

public static double[][] transpose(double[][] M) {

// find number of rows and columns in matrix M

int n = M.length, m = M[0].length;

System.err.println("Data to put in a transposition: " + M);

// create empty transpose matrix of size m\*n

double M\_transpose[][] = new double[m][n];

// traverse matrix M

for (int i = 0; i < n; i++) {

for (int j = 0; j < m; j++) {

//assign M\_transpose[j][i] as M[i][j]

M\_transpose[j][i] = M[i][j];

}

}

System.err.println("Transposition : " + M\_transpose);

return M\_transpose;

}

}

Example of the Transpose method adapted for use with an ArrayList Object

and the dot product function detailed below which also might need adjusting.

public class Dot {

public static double dot(double[][] inputs, double[] inputsB) {

System.err.println("Starting to calculate the dot product...");

double sum = 0;

System.out.println(inputs);

System.out.println(inputsB);

for (int i = 0; i < inputsB.length; i++) {

System.err.println("The inputsA length is same as the inputsB length");

sum += inputs[i][i] \* inputsB[i];

}

System.err.println("Dot product sum is: " + sum);

return sum;}}

Example of the dot product method modified for use with ArrayList Objects

The nature of the training module has been modified so far to accept only the training data to alleviate the necessity to provide other arguments to the function making it easier to activate the Neuron later. We do that because it becomes increasingly easier to call the functions within the volume. After all, we are attempting to send the synaptic data as the input to other Neurons effectively. What we really need is to modify the training function (Neuron) to call the training data from within and accept only synaptic data as an argument and as an output once trained.

public class Training {

static HashMap<Integer, ArrayList<Object>> beehive = NeuralNet.getBeehive();

static double learningrate = NeuralNet.getLearningrate();

static double [] hiddenLayerWeights = NeuralNet.getHiddenLayerWeights();

static double [] hiddenLayerOutputs = NeuralNet.getHiddenLayerWeights();

static double [] outputLayerWeights = NeuralNet.getHiddenLayerWeights();

static double [] outputLayerOutputs = outputLayerWeights;

public static Object training(int ID) throws IOException {

System.err.println("Neuron Training ID::" + ID);

double[][] input = NeuralNet.getInputs();

double[] output = NeuralNet.getDesiredOutput();

int t\_qty = NeuralNet.getT\_qty();

double sum = 0;

for (int i = 0; i <= t\_qty; i++) {

// Calculate the error (The difference between the desired output

// and the predicted output).

for (int in = 0; in <= input.length-1; in++) {

Error.error(input[in], outputLayerOutputs, output, hiddenLayerOutputs, outputLayerWeights, learningrate, hiddenLayerWeights);

}

for (int s = 0; s <= input.length-1; s ++) { sum = Summation.summation(output, outputLayerWeights);}

}

sum -= (sum + sum);

System.out.println("The summation of accuracy was: %" + sum);

return Training.class;}}

Example of the training method also described and used as the Nucleus of a Neuron

What we have is the training module being a part of the thinking aspect of the Neuron which is deployed inside the volume at random when called. As you can see from our training module (nucleus) we can run call backs to other math functions and operations for better processing.

## Volume - Neuron

The Neuron is simply an Object of multiple functionalities, it possesses the capability of outputting and making use of it’ proximities for connectivity on call from external function calls and has the capability of having the nucleus (The training method or Mathematical algorithm) called externally to become in an active state, taking in information and outputting data. These Neurons will have to be modified further to accept only the synaptic data and make a call to the training\_set internally.

public class Neuron {

//public static final ArrayList<Object> neuron = neuron(null);

/\*\*

\* Thinking if we input the connections to think() as well as the

\* synaptic weights from the sigmoids, we will have inputs as outputs

\* and vise versa, since we need the synapses to receive the output of

\* each neuron to make use of as an input because we are working on a

\* multi-neuron system which are all interconnected according to proximity

\*

\* Though there will be dormant neurons if too large a volume (it's fine)

\* We still need all the neurons to talk according to their connectivity.

\* @throws IOException

\*/

public static double a() {

return 0;

}

public static ArrayList<Object> neuron () throws IOException {

ArrayList<Object> neuron = new ArrayList<>();

neuron.add(a()); //Training module or sigmoid etc etc, initially empty method, new class method added in HiveNest mapping.

neuron.add(Touch.t()); //Mating zone, the quintessential zone of mating, energy transfer, bioelectrical transfer, atomic emf/emp bonds

neuron.add(Green.g()); //emf

neuron.add(Red.r()); //red zone, widest radii

neuron.add(SurfaceSphere.surface(Touch.getR())); // touch volume surface area

neuron.add(SurfaceSphere.surface(Green.getR())); // green surface area

neuron.add(SurfaceSphere.surface(Red.getR())); // red surface area

neuron.add(0); // I/O weights

neuron.add(0); // x

neuron.add(0); // z

neuron.add(0); // y

return neuron;}}

Demonstrating the Neuron in it’ entirety and the method calls embedded to the memory space

The way the Neuron works in the example as stated in the first iteration’ documentation, the Methods are packed into an ArrayList Index memory space and can be called into functionality or ‘activation’ by accessing the index in the standard way even through hash mapping. It is the same method by which a method call-back works except the data is stored into memory allocated with an index, similar in nature to the stack without the first in first out rules.

The method disclosed here permits for whole methods, various data types and even void functions which simply perform an action. In doing so, we can organise specific connections to specific neurons or even have a single input neuron in a volume (lobe) and an output neuron hard coded into the lobe and output into a secondary lobe for further processing. The potential to explore the two hemispheres is seemingly possible with the current model proposed and we may be able to look at mapping individual cell signals to the indices of the volumes based on current research today [SciTechDaily, 2023].

As such after gathering plenty of data and analytics, we have a wide range of potential for further development and research and again, once those optimums are found, we can port to a lower-level language and run additional tests and benchmarking in anticipation for embedding into chipsets.

## Volume - Mapping

Since we have the training module as part of the nucleus and call the training set directly from perhaps a plugin method. We could possibly change the nucleus all together simply by using a stack of differing artificial intelligences to be deployed in the nucleus. The deployment is simply pushed in by allocating the Neuron to the mapping function with the correct arguments such as the cubic size of the volume and the Neuron quantity to accompany the mapping functionality in cubic mapping class and mapped using the populate function below.

/\*\*

\* Randomly populate the mapped volume with Neurons

\* @param neuron

\* @param qty - Quantity of neurons

\* @param v - Received autonomously from mapping()

\* @throws IOException

\*/

public static void populate(ArrayList<Object> neuron, int qty, double v) throws IOException {

System.err.println("Populating volume with Neurons...");

Random rand = new Random();

for (int i = 0; i <= qty; i++) { int m = rand.nextInt((int) v); beehive.put(m, Neuron.neuron());}

System.err.print("Original mapping hashcode after first generation: " + beehive.hashCode() + "\n");

NeuralNet.setBeehive(beehive);}

Demonstration of the volume being populated at random indices

# Connectivity

When considering the connectivity of Neurons in our Neural Network, how best to do that? We have each Neuron with a surface area, a volume and x,z,y coordinates in our slightly unorthodox system. Typically, a Neural Network would consist of the perceptron model, not dissimilar to ours.

The perceptron model is ‘a typical and very well known and used model in neural networking’, it is easily replicated and represented in computer code and even mathematically [Vadapalli, 2020]. Our model is a perceptron based like many others though it is specifically though there is one difference; We are using collision theory in its connectivity as opposed to Minimum Spanning Trees (MST) such as Primm, Kruskal and Dijkstra [Sedgewick and Wayne, 2011] and in itself, presents more complexity.

That is not to say we could not implement the MST as an autonomous path tracer through each connected Neuron. In example, if each Neuron were connected in proximity based connections, we could pick a neuron and use MST to traverse from start point to end node point to get an outcome. For the first iteration, we’ll work without MST though.

Looking at the Radial Basis Network (RBN) descriptions in DeepAi and SimpliLearn [2023], the notion of the network being an RBN is possible however at this point we are focussing on making the proximity connectivity and communication work before moving into the type of training model we are going to use. At present we are simply using a gradient descent model in the training as described by Trehan [2020].

## Connectivity - Volume

One of the first issues encountered is what sort of volume do we give the Neurons in their proximities? We could set a generics 1, 2, 3. However, in previous tests that did not work because the volume of the container of our HashMap had not been linked to the proximities in any way, thus we never had any connections even though the math was correct in logic [MDN Web Docs, Jackson, StackExchange].

As such we have linked in the volumes after making the proximities inclusive in the factorisation of the volume itself within the radii. In example, because one Neuron is housed in one Index, we want to say ‘for the closest radius, I want to talk to my closest cells’ in doing so, because we are working with cubic logic in the cells, we want to say one nineth of the volume as demonstrated below.

/\*\*

\* Using Cubic logic, assign a proximity based on a radius

\* @author xer0n3

\*

\*/

public class Touch {

/\*\*

\* return the spherical volume of touch zone

\* based on the cubic logic

\*/

private static double r = (Math.pow(NeuralNet.getV\_qty(), 3)) / 9;

public static double getR() {

return r;

}

public static void setR(double r) {

Touch.r = r;

}

/\*\*

\* Send out the volume on call to index 1 of neuron

\* or call directly

\* @return volume

\*/

public static Object t() {return Radial\_volume.a((float) Touch.getR());}}

Example of the radius, proximity and integration of portioning of the volume created.

After doing so, we can call the radius as a portion of the volume demonstrated in our class below and with that in mind the port from JavaScript went well from MDN Web Docs [2023] and after cycling through the index of the volume, we are able to make use of the port over to Java to ensure connections were made.

public class PointInside {

static HashMap<Integer, ArrayList<Object>> hive = NeuralNet.getBeehive();

public static int[] is(int key) {

int[] connected = new int[(int) Math.pow(NeuralNet.getV\_qty(), 3)];

int bkey = key;

//get our keys (key b) coordinate values

int bees\_x = GetX.getx(bkey);

int bees\_z = GetZ.getz(bkey);

int bees\_y = GetY.gety(bkey);

for (Entry<Integer, ArrayList<Object>> bee: hive.entrySet()) {

//Sphere location

int x = (int) bee.getValue().get(8);

int z = (int) bee.getValue().get(9);

int y = (int) bee.getValue().get(10);

double distance = Math.sqrt((bees\_x - x) \* (bees\_x - x) + (bees\_z - z) \* (bees\_z - z) + (bees\_y - y) \* (bees\_y - y));

for (int i : connected) {

if (distance < Red.getR()) {

connected[i] = bee.getKey();

}

if (distance == Green.getR()){

connected[i] = bee.getKey();

}

if (distance < Touch.getR()) {

connected[i] = bee.getKey();

} else {

break;

}

}

}

return connected;}}

Example of the collision detection ported from JavaScript to Java MDN Web Docs [2023].

To send through the key value to be assessed we needed the integer value and a call back to get the connections stored however, we need some randomeness to ensure that not all neurons are connected necessarily because it makes for interesting results.

//Populate the index of connect with the initial keys and change the values to connections

for (Entry<Integer, ArrayList<Object>> bee : hive.entrySet()) {

for (int p : connect[bee.getKey().intValue()]) {

int[] a = PointInside.is(bee.getKey());

connections.put(bee.getKey(), a[p]);

}}

Example of each of our Neuron being sent to PointInside.is() for connectivity assessment.

## Connectivity – Results

In doing so, we end up having multiple clusters and linking demonstrated in the results of the connectivity ArrayList, permitting us to access the Neurons by Key.

[6, 8, 9, 10, 11, 12, 13, 13, 13, 16, 17, 17, 17, 27, 27, 31, 32, 32, 38, 38, 41, 41, 43, 43, 50, 54, 58, 59, 59]

[2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 16, 17, 24, 26, 27, 31, 32, 34, 35, 37, 38, 41, 43, 48, 50, 54, 58, 59]

{2=6, 4=8, 5=9, 6=10, 7=11, 8=12, 9=13, 10=13, 11=13, 12=16, 13=17, 16=17, 17=17, 24=27, 26=27, 27=31, 31=32, 32=32, 34=38, 35=38, 37=41, 38=41, 41=43, 43=43, 48=50, 50=54, 54=58, 58=59, 59=59}

Example data from the randomised Neuron distribution after collected, we can see variances here in each sample below

[4, 8, 11, 11, 11, 22, 26, 27, 28, 28, 28, 28, 32, 32, 37, 41, 44, 46, 48, 49, 52, 53, 53, 57, 58, 62, 62]

[2, 4, 8, 10, 11, 18, 22, 23, 24, 25, 26, 27, 28, 32, 36, 37, 41, 43, 44, 46, 48, 49, 52, 53, 57, 58, 62]

{**2=4, 4=8, 8=11, 10=11, 11=11**, **18=22**, 22=26, 23=27, **24=28, 25=28, 26=28, 27=28, 28=32, 32=32**, 36=37, 37=41, 41=44, 43=46, 44=48, 46=49, 48=52, 49=53, 52=53, 53=57, 57=58, 58=62, 62=62}

Example of linking detected between neurons highlighted in bold green.

[2, 5, 9, 10, 13, 14, 17, 18, 19, 19, 22, 22, 25, 25, 25, 30, 44, 44, 48, 52, 52, 54, 54, 54, 54, 63, 63]

[0, 2, 5, 8, 9, 10, 13, 14, 15, 17, 18, 19, 22, 24, 25, 30, 41, 42, 44, 48, 49, 50, 51, 52, 54, 61, 63]

{0=2, 2=5, 5=9, 8=10, 9=13, 10=14, 13=17, 14=18, 15=19, 17=19, **18=22, 19=22, 22=25, 24=25, 25=25**, 30=30, **41=44, 42=44, 44=48, 48=52, 49=52**, **50=54, 51=54, 52=54, 54=54, 61=63, 63=63**}

Examples of Clustering in connectivity ArrayList.

# Gathering

We are working on artificial neurological systems and thus, it would be nice if we could see what the individual Neurons are doing and even what the randomised structure of the volume looks like when Neurons have been added to the volume.

From that data collection we can assimilate optimums once data analysis has occurred, simply saying 100 billion Neurons [Spencer-Harper, 2015] processing n = n\*(x + y) / (pow(n, 4)) \* n is a little dull because in essence Neurons are adjusting their sigmoid functionality because of the nature of bioelectrical and biochemical stimulus affecting them. As such each lobe of the brain, performs a slightly different job and varies slightly in individuals and species [Anatomy, 2022].

To gather the data required to assess what constitutes to a nano sample of Neurons we need both logs piped out to the error console since that data is collected intrinsically and stored locally, in addition to file storage for multiple append calls to the file to gather sets of data for analysis. The collection process also enables timers to be written in later for benchmarking (though write outs to file, error logs and timers will slow the performance somewhat).

## Gathering – Data and Logs

When gathering data, we are going to need to autonomously check and or create our file and to access it for writing data. In addition, what data do we collect? What is important? Well, we already discussed the mapped volume and its randomised configuration.

We would want the connections detailed to assess whether dormant Neurons were created (too far away from any proximities) and therefore not sending or receiving data to or from other Neurons. We would also want to know what data was being sent between the Neurons, what the synaptic weights were for firing off to the surrounding connected Neurons.

We may even want to simply gather reports of what is happening during the process of running the tests including benchmarking. These pieces of data are valuable during the testing stages and phases. Moreover, which tools would be the simplest way to gather such complex data?

The inbuilt system input output stream will be best suited for the error outputs since they are already built into and accessible by our IDE. However, the data file storage read and write will need to be automated somewhat.

## Gathering – Finite State Machine Triggers

In doing so and in keeping with the autonomous collection of varying data, how best to do that? Well, there is the finite state machine. The finite state machine could be used to change the state of a variable and trigger a response of some kind in a separate method or class for dealing with data change.

public enum *PrinterState* {

***VOID***,

***CONN***,

***TOUCH***,

***GREEN***,

***RED***,

***MAVOL***,

***SYNAP***,

***TRANSB4***,

***TRANSFTR***,

***SITUATION***,

***ADJUSTMENT***,

***TRAINSET***,

***ERRORS***,

***TIME***;

}

Example of the finite state machine to be used

In our example, the change is a switch statement after a variable has been set to print the variable’ data to file. We set the new variable inside the printer class using encapsulation,

error.add(((output.size()-1 - sum) / 100) \* output.size()-1);

Logger\_Writer.*setErrors*(error);

Example of how the data is set in the log printer

From there we call to trigger the state change,

Logger\_Writer.Logger\_Printer(PrinterState.ERRORS);

Demonstration of how the event is triggered in the log writer via the finite state machine

And in the printer module of the Logger Writer class the event is triggered causing the variable to be printed to file,

case ***ERRORS***:

try {

*Logger\_Checker*();

stack.write("Errors before computation of synapse:\n" + Logger\_Writer.*getErrors*() + "\n");}

catch (IOException e) {

System.***err***.println("An error occurred." + e);

e.printStackTrace ();

}

break;

The switch statement which is triggered by the event from the finite state machine

These techniques will permit storage of the data and it’ structure into a file which has been created by the system in place autonomously and checked for readability and write access.

public static void Logger\_Creator() {

try {

File data\_doc = new File("data\_doc.txt");

if (data\_doc.createNewFile()) {

System.***err***.println("File " + data\_doc.getAbsolutePath() + " " + data\_doc.getName() + " created.\n");

} else if (data\_doc.exists()) {

System.***err***.println("File " + data\_doc.getAbsolutePath() + " " + data\_doc.getName() + " exists, nothing done.\n");

}

} catch (Exception e) {

System.***err***.println(e);

}

*Logger\_Checker*();

Example of the logger creating a file and running a check on it’ capabilities

The Logger creator can also be called autonomously during the checking process requested through each trigger from the state machine, as can the creator from the checker.

public static boolean Logger\_Checker() {

int count = 0;

if (!*data\_doc*.exists()) {

*Logger\_Creator*();

}

while (*data\_doc*.exists()) {

System.***err***.println("Can output file : " + *data\_doc*.getName() + " is located in directory: " + *data\_doc*.getAbsolutePath() + ".\n");

if (*data\_doc*.canWrite()) {

System.***err***.println("Can write to output file : " + *data\_doc*.getName() + ".\n");

count += 1;

} else {

System.***err***.println("Cannot write to output file : " + *data\_doc*.getName() + ".\n");

return false;

}

if (*data\_doc*.canRead()) {

System.***err***.println("Can read from output file : " + *data\_doc*.getName() + ".\n");

count += 1;

} else {

System.***err***.println("Cannot read from output file : " + *data\_doc*.getName() + ".\n");

return false;

}

}

if (count == 2) {

return true;

} else {

System.***err***.println("Errors were encountered when trying to access and recreate " + *data\_doc*.getAbsolutePath() + ".\n");

System.***err***.println("Read back through error reports.\n");

return false;

}

}

Example of checking the file’ permissions and existence

# Interface

The interface to activation and logging the process which is going to follow on from the running of the code has to trigger the logging, kick off the connectivity and the randomisation of the synaptic weights and then begin to process the loading into the mapped volume.

*@SuppressWarnings*("unchecked")

public static ArrayList<ArrayList<Object>>deploy () throws IOException {

Synaptics.*setNeuron*(*neuron\_qty*);

Logger\_Writer.*setSynaptics*(*synaptics*);

Logger\_Writer.*setSituation*(*situation*);

Logger\_Writer.Logger\_Printer(*PrinterState*.***VOID***);

Logger\_Writer.Logger\_Printer(*PrinterState*.***SYNAP***);

Logger\_Writer.Logger\_Printer(*PrinterState*.***SITUATION***);

Map<Integer, ArrayList<Object>> volume = Mapped\_volume.mapped\_volume(*neuron\_qty*, *vol*);

Logger\_Writer.*setMapped\_volume*(volume);

Logger\_Writer.Logger\_Printer(*PrinterState*.***MAVOL***);

ArrayList<Object> connections = new ArrayList<>(Connections.connections(volume));

Logger\_Writer.*setConnectionss*(connections);

Logger\_Writer.Logger\_Printer(*PrinterState*.***CONN***);

ArrayList<ArrayList<Object>> data = new ArrayList<>();

for (Entry<Integer, ArrayList<Object>> be : volume.entrySet()) {

System.***err***.println("Volume Key: " + be.getKey() + " Volume Value: " + be.getValue() + ".\n");

}

System.***err***.print("HashCode : " + volume.hashCode());

System.***err***.println("The neuron Matrices are: \n" + volume);

System.***err***.println("The Neurological connections are: \n" + connections);

data.add((ArrayList<Object>) connections);

data.add((ArrayList<Object>) volume);

Logger\_Writer.*Logger\_Generic*("Connections and Neurologically Mapped Volume respectively: \n " + data.get(0) + "\n" + data.get(1) + "\n");

System.***err***.println("Complete data stream from connections and volume mapped: " + data + ".\n");

//Activate the training module in each neuron

for (Entry<Integer, ArrayList<Object>> activate : volume.entrySet()) {

ArrayList<Object> neurological = new ArrayList<>(activate.getValue());

for (int a = 0; a <= neurological.size(); a++) {

neurological.get(0);

System.***err***.println("Neuron: " + activate.getKey() + " - Activated!\n");

}

}

System.***err***.println("Activated Neurons.\n");

Logger\_Writer.*Logger\_Generic*("The mapped volume and it's connections are:\n" + data);

return data;}

The interface used to interact with the whole composite of the Artificial Intelligence

The data returned is a multi-dimensional ArrayList of the volume and it’ related connections. Doing so permits us to store the data and output into the error console, as such we can check and validate whether there are potential flaws or even dormant Neurons in the volume furthering our assessment for optimal placement of Neurons.

# The Brain Introduction

To really get an idea of what we are seeking to engineer, we must take a brief look at some of the Human Brain functionality and biological makeup, to include it’ aggregate roles within the cognition and behaviour of individuals.

“

* The frontal lobe is located behind the forehead, and is responsible for considerable of the complex cognitive function: Reasoning, imagination, planning, values and behavior.
* The parietal lobe is located in the upper back of the frontal lobe. It covers the sensitive cortex (processing those messages related to touch, palate and body temperature), and the motor cortex (controlling the movement).
* The temporal lobe is located behind the temple, sheltering the auditory cortex, taking care of the language comprehension, and acting over emotions and memory.
* The occipital lobe is located behind the head, controlling the visual cortex in charge of handing out what the individual sees.

”

Excerpt from Anatomy.co.uk [2022]

When considering the four lobes there is multiple functionalities to consider and how they are going to interact and pass information to and from each other. Each lobe serves a different purpose to both he body and cognition including emotions. Figure 1, demonstrates a basic overview of the complexity of the Human Brain and some of it’ functionality in the hemispheres.

Diagram

Description automatically generated

Figure 1. The Human Brain and some of it’ functionality in the hemispheres

## The Brain – sector use-cases

The diencephalon is considered the ‘core’ of the brain in combination with the Thalamus, which takes information from the body and several sensory organs and filters the information for transferring to the cerebral cortex to mitigates Brain overload. In addition, the Hypothalamus which deals with monitoring vital functions and ‘components’ [Anatomy, 2002]. These neurological functions can be replicated and transferred to machine code fairly easily these days, in particular where mechanical and software engineering is concerned. However, our primary focus is, identifying key areas of the Brain and replicating the neurological signal in the desire to achieve the relevant cognitive processing inside machines.

Though most of the work has already been done, the task seems to be more of a challenge on the path to understanding the processes and their functionality for Artificial representation in machines, hence the project we are committing to in the future.

We are taking a look at individual and cluster Neuron behaviour including mapping them with unique identifiers, permitting each cell to output to log files and providing feedback into the native log system as the events of the calculus unfold.

# Next Iteration

We have developed a naïve representation of a very small composite of Neurons which behave just as and whilst making use of one test bed which has been ported and modified to the use case. Perhaps the next iteration will make use of the volumetrics as lobes each housing different Ai for their behaviour and for the lobe to have a single joining Neuron to a central volume which handles and makes use of the information received known as the ‘core lobe’ or diencephalon.

“

The third-largest part is the diencephalon, located in the core of the brain. A complex of structures roughly the size of an apricot, its two major sections are the thalamus and hypothalamus. The thalamus acts as a relay station for incoming nerve impulses from around the body that are then forwarded to the appropriate brain region for processing. The hypothalamus controls hormone secretions from the nearby pituitary gland.

”

Excerpt from National Geographic [2023]

A graphical user interface would be a suitable interface to the work for ease of use and further testing and development making it easier to apply, deploy and gather results faster. A secondary system will need to be developed for analysis of the data being output to the error console and the storage file.

Further developments in the next iteration could include being able to change the training in a single cortex and perhaps even the data being fed in via sockets or even a simple image and collection of images. It would be a good idea to test other existing Ai and benchmark them. Perhaps later after sufficient testing, sockets and a network feed to assess data input and output, self-assessment and even image-based processing to generate artwork or to assist with facial recognition, all being well of course. It would be good to see if creating the two hemispheres, one logical and one creative either side of the volumes or to map their functionality according to anatomic reference.

Whilst some tasks specifically the frontal lobes, which are connected to executive functions such as reasoning and scheduling, are already replicated in software or firmware in modern day computers with logic and background tasks and scheduling; Abstract thinking and self-control are only partially implemented and are well worth a look at replicating since abstract thinking possibly making use of both hemispheres. After all, we’re trying to build a brain, not a toy.

# Programming Code

The programming code for the project can be found at – <https://www.github.com/russc-xer0n3/II>II

The first iteration and documentation can be found at – <https://www.github.com/russc-xer0n3/I>

# References

Anatomy., 2022. Human Brain. *Introduction to the Human Brain* [online] Available from: <https://anatomy.co.uk/brain/> [06/01/2023].

Baeldung., 2022., Java Multi-dimensional ArrayList [online] Available from: <https://www.baeldung.com/java-multi-dimensional-arraylist> [20/11/2022].

Baeldung., 2022. Implementing Simple State Machines with Java Enums [online] Available from: <https://www.baeldung.com/java-enum-simple-state-machine> [01/01/2023].

ClevelandClinic., 2023. Cerebral Cortex [online] Available from: <https://my.clevelandclinic.org/health/articles/23073-cerebral-cortex> [06/01/2023].

DeepAi., 2023. Radial Basis Functions [online] Available from: <https://deepai.org/machine-learning-glossary-and-terms/radial-basis-function> [28/01/2023].

Geeks for Geeks., 2022. File createNewFile() method in Java with Examples [online] Available from: <https://www.geeksforgeeks.org/file-createnewfile-method-in-java-with-examples/> [01/01/2023].

Geeks for Geeks., 2021. Java Program to Implement ArrayList API [online] Available from: <https://www.geeksforgeeks.org/java-program-to-implement-arraylist-api/> [20/12/2022].

Geeks for Geeks., 2022. Java.util.HashMap in Java with examples [online] Available from: <https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/> [20/11/2022].

Jackson., M. 2002. Pool Hall Lessons: *Fast, Accurate Collision Detection Between Circles or Spheres* [online] Available from: <https://www.gamedeveloper.com/programming/pool-hall-lessons-fast-accurate-collision-detection-between-circles-or-spheres> [17/01/2023].

JavaTpoint., 2022. Java Map Interface [online] Available from: <https://www.javatpoint.com/java-map> [20/11/2022].

JavaTPoint., 2022. Java program to Transpose Matrix [online] Available from: <https://www.javatpoint.com/java-program-to-transpose-matrix> [29/11/2022].

Kareem, I. D., 2016. How to make my dot product method in Java faster or more efficient? [online] Available from: <https://stackoverflow.com/questions/40434418/how-to-make-my-dot-product-method-in-java-faster-or-more-efficient#:~:text=I%20have%20a%20small%20Java%20method%20that%20is,%5Bi%5D%3B%20%7D%20outputs%20%5Bj%5D%20%3D%20sum%3B%20%7D%20%7D> [29/11/2022].

KenHub., 2023. Cerebral Cortex [online] Available from: <https://www.kenhub.com/en/library/anatomy/cerebral-cortex> [06/01/2023].

Lewis. J., Loftus. W., 2012. Java Software Solutions Seventh Ed. Foundations of program design. Addison Wesley, Boston, MA.

MDN Web Docs., 2023. 3D Collision detection [online] Available from: <https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection> [17/01/2023].

Medical News Today., 2023. What to know about the Brain [online] Available from: <https://www.medicalnewstoday.com/articles/brain> [06/01/2023].

National Geographic., 2023. The Human Brain, explained*. Learn about the most complex organ in the human body, from its structure to its most common disorders.* [online] Available from: <https://www.nationalgeographic.com/science/article/brain-2/> [06/01/2023].

Oracle., 2022. ArrayList [online] Available from: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html> [20/12/2022].

Oracle., 2022. forEach-java.util.function.Consumer [online] Available from: <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#forEach-java.util.function.Consumer-> [20/12/2022].

Oracle., 2022 Math [online] Available from: <https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html> [20/12/2022].

SciTechDaily., 2023. Biology News. *Scientists Have Created the Most Detailed Map of the Brain’s Memory Hub – And It Could Change Our Understanding of Memory* [online] Available from: <https://scitechdaily.com/scientists-have-created-the-most-detailed-map-of-the-brains-memory-hub-and-it-could-change-our-understanding-of-memory/> [06/01/2023].

Sedgewick. R., Wayne. K., Et. al. 2011. Algorithms Fourth Ed. Addison Wesley, Princeton University, U.S. NJ.

1. SimpliLearn., 2023. What are Radial Basis Functions Neural Networks? Everything You Need to Know [online] Available from: <https://www.simplilearn.com/tutorials/machine-learning-tutorial/what-are-radial-basis-functions-neural-networks#:~:text=Radial%20Basis%20Function%20%28RBF%29%20Networks%20are%20a%20particular,three-layer%20architecture%2C%20universal%20approximation%2C%20and%20faster%20learning%20speed>. [28/01/2023].
   1. Spencer-Harper., M. 2015. How to build a simple neural network in 9 lines of Python code [online] Available from: <https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1> [29/11/2022].
   2. StackExchange., 2011. Game Development. *Sphere collision response* [online] Available from: <https://gamedev.stackexchange.com/questions/16976/sphere-collision-response> [17/01/2023].

StackOverflow., 2015. Transpose ArrayList<ArrayList<String>> in Java [online] Available from: <https://stackoverflow.com/questions/28057683/transpose-arraylistarrayliststring-in-java> [29/11/2022].

Trehan., D. 2020. Gradient Descent Explained. *A comprehensive guide to Gradient Descent* [online] Available from: <https://towardsdatascience.com/gradient-descent-explained-9b953fc0d2c> [27/01/2023].

TutorialsPoint., 2022. Java.io.File.list() Method [online] Available from: <https://www.tutorialspoint.com/java/io/file_list.htm#:~:text=Java.io.File.list%20%28%29%20Method%201%20Description%20The%20java.io.File.list%20%28%29,abstract%20pathname.%205%20Exception%20...%206%20Example%20> [01/01/2023].

1. Vadapalli,. P. 2020. 7 Types of Neural Networks in Artificial Intelligence Explained [online] Available from: <https://www.upgrad.com/blog/types-of-neural-networks/#:~:text=Types%20of%20Neural%20Networks%201%20Perceptron%20The%20Perceptron,Networks%20...%207%20Long%20Short-Term%20Memory%20Networks%20> [26/01/2023].

W3Schools., 2022. Java Class Attributes [online] Available from: <https://www.w3schools.com/java/java_class_attributes.asp> [30/12/2022].

W3Schools., 2022. Java Classes [online] Available from: <https://www.w3schools.com/java/java_classes.asp> [30/12/2022].

W3Schools., 2022. Java Encapsulation [online] Available from: <https://www.w3schools.com/java/java_encapsulation.asp> [30/12/2022].

W3Schools. 2022. Java Enums [online] Available from: <https://www.w3schools.com/java/java_enums.asp> [01/01/2023].

W3Schools., 2022. Java File Create [online] Available from: <https://www.w3schools.com/java/java_files_create.asp> [01/01/2023].

W3Schools., 2022. Java Files [online] Available from: <https://www.w3schools.com/java/java_files.asp> [01/01/2023].

W3Schools., 2022. Java HashMap [online] Available from: <https://www.w3schools.com/java/java_hashmap.asp> [20/11/2022].

W3Schools., 2022. Java OOP [online] Available from: <https://www.w3schools.com/java/java_oop.asp> [30/12/2022].