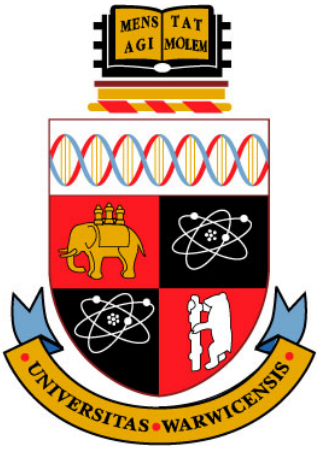

Hyper-graphical extensions of randomly grown neural networks

MA4J5 Structures of Complex Systems

Ramón Nartallo-Kaluarachchi

Lecturer: Dr. Markus Kirkilionis

Warwick Mathematics Institute

University of Warwick

2021-2022

Abstract

There is a strong link between network models and computational neuroscience, particularly machine learning and neural networks. Originally, neural networks took direct inspiration from the human brain in terms of both structure and function. However, it is common to see network topologies in the artificial neural networks used in machine learning that are neither explainable nor motivated. Similarly, in computational neuroscience and spiking neural networks, random topologies are common. In 2015, Agazi et al presented a simple branching process model of the growth of a neuron as well as a notion of connectivity between these single neurons [1]. This allows for the stochastic simulation of small networks that mimic the structure of biologically grown neural networks. In this project, the aim was to recreate the model and then to significantly expand to hyper-graphical extensions and useful applications. The extensions include: excitatory/inhibitory networks, clustered networks of local and projection neurons, layered neural networks for deep learning and Hopfield networks. With these extensions and the associated algorithms, the model can be used to generate a wide range of biologically motivated network topologies ready for application in computational neuroscience and machine learning.

Keywords: Random graphs, neural networks, hyper-graphs, brain growth, graph theory, branching processes

Contents

1 Introduction

Within both machine learning and computational neuroscience, there is a sub field focused on building biologically motivated artificial neural networks that mimic the function of the human brain. However, even these networks typically neglect network topology and structure as a important biological feature, often opting for random or arbitrary connectivity. Random graph theory is a useful tool in studying both the structure and dynamics of biological networks, including those in the human brain. The neural networks found in the human brain are, of course, the product of biological growth processes and formation over time. An interesting approach to the problem of producing biologically motivated and plausible connectivity in networks is emulating the dynamics of biological neural growth.

A complex, established tool for the simulation of such networks is NETMORPH presented by Koene et al. [2]. Similarly to the model discussed here, neurons are modelled as rooted binary trees governed by a branching process. The branching rate in this state-of-the-art model is assumed to be 'a monotonically decreasing function of time' [2]. In this project, the model used was presented by Ajazi et al. [1]. It is a simplification of the NETMORPH model that assumes a constant branching rate for all times. The model is also restricted to 2D. Whilst this is sufficient for the majority of applications and extensions discussed in this project; the model, and the extensions presented here, could be generalised to 3D.

A key aim of this project was to take the graphical nature of the model and extend it to hyper-graphical structures. By doing this, one can mimic hierarchical features of the brain as well as model diverse types of neurons, thus creating more biologically plausible and rich networks. The hyper-graphical extensions considered in this project are: networks of excitatory-inhibitory neurons; networks of local and projection neurons, organised into clusters; layered, feed-forward networks, such as those used in deep learning; and Hopfield networks.

This paper should be read alongside the Jupyter notebook that allows for the simulations of these networks. The notebook is available at `https://github.com/rnartallo/randomlygrownnetworks`. The code is written in Python3.

2 A model of a single neuron

The model presented here was first presented Ajazi et al. (2015) [1]. It is a direct simplification of the model presented in Acimovíc et al. (2011) [3].

A *neuron* is assumed to be, at any time, a rooted, random tree. We call the root the *soma* of the neuron. The branches of the tree represent the growing *axons* of the neuron.

We denote our time parameter by $t \geq 0$. At time $t = 0$, the neuron is represented by a single point $s \in \mathbb{R}^2$, the soma. At any time t, the neuron is represented by the random tree $N_s(t) \subset \mathbb{R}^2$. The neural growth is governed by the two processes of elongation and branching of axons.

At $t = 0$, a segment begins to grown from s in a random direction between two parameters: the angle upper-bound, θ_U; and the angle lower-bound, θ_L. The segment grows with constant speed, set to 1. This is not a restriction as time can be rescaled. At a random time τ_0, governed by an exponential distribution with mean $1/\lambda$, the segments splits into two new segments with random directions between the bounds. These develop independently in the same manner. Each of these segments will split at times $\tau_0 + \tau_1$ and $\tau_0 + \tau_2$ respectively. The τ_i are independently, identically distributed from the same exponential distribution with mean $1/\lambda$. This process repeats whilst $t < T$; where T is a parameter representing the total time given for growth.

The parameters for the model are:

- s - the start point of the neuron and the position of the soma.
- θ_U - the angle upper-bound for the direction of an axon.
- θ_L - the angle lower-bound for the direction of an axon.
- λ - the rate/inverse scale of the exponential distribution governing branching times.
- T - the stopping time for the model.

In the majority of cases we consider $\theta_U = -\theta_L$.

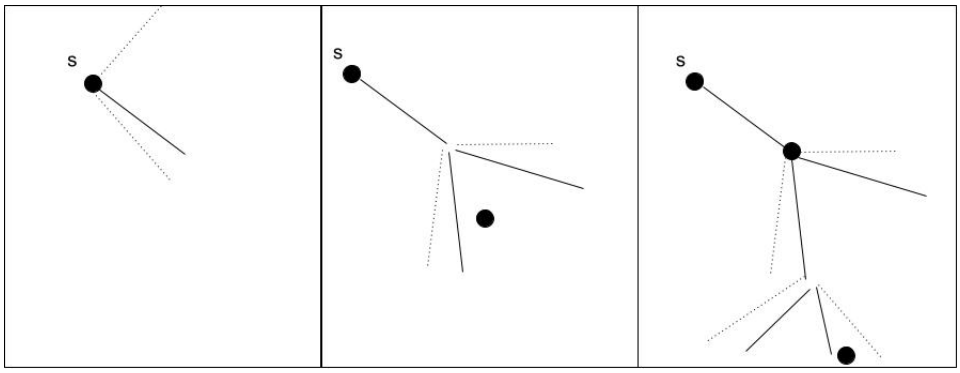

Figure 1: Model of a growing neuron

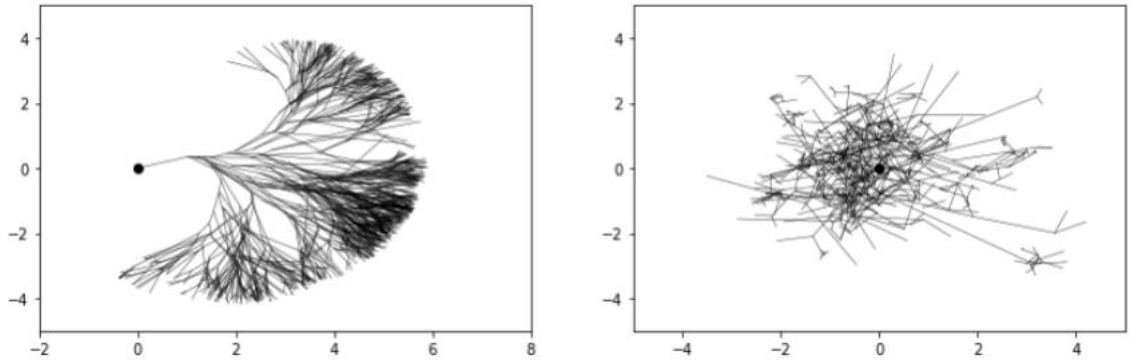

Figure 2: Simulated neurons: with angles in $[-\pi/6, \pi/6]$ and $[-\pi, \pi]$; $\lambda = 1$; $T = 6$

2.1 Algorithm for modelling a single neuron

Using the model and the parameters, we can define the following algorithm for modelling a single neuron.

- Parameters: $s, \theta_U, \theta_L, \lambda, T$
  - $N_s(0) = \{s\}$
  - Generate a random direction $d \sim \mathrm{U}(\theta_L, \theta_U)$
  - **while** $t < T$
    - Generate a branching time $\tau \sim \mathrm{Exp}(1/n\lambda)$ where n is the number of growing axons
    - Randomly select one of the growing axons (uniformly)
    - Generate two new directions for the new segments $d_1, d_2 \sim \mathrm{U}(\theta_L, \theta_U)$
    - $t = t + \tau$

As seen in the simulated neuronal trees in Figure 2, we can now simulate the growth of individual neurons on the plane.

3 A network level model

3.1 Connectivity

In order to construct networks from this single neuron model, Ajazi et al. define a notion of connectivity for two neurons grown on the same plane [1].

Firstly, we define a notion of distance between the tree of a neuron starting at s, $N_s(t)$, and a point ω.

Definition .1. Let $N_s(t)$ represent the axonal tree of a neuron starting at s after time t. Let $\omega \in \mathbb{R}^2$. The *distance* between $N_s(t)$ and ω is denoted by $d(N_s(t), \omega)$ where

$$d(N_s(t), \omega) := \min\{||x - \omega|| : x \in N_s(t)\} \tag{3.1}$$

where $||\cdot||$ represents the standard Euclidean distance in $\mathbb{R}^2$

With this notion of distance, we now introduce a new parameter, r, called the *radius*. We now define our notion of directed connectivity.

Definition .2. Let $N_s(t), N_\omega(t)$ represent the axonal trees of neurons S, Ω starting at s, ω after time t. Let r be the radius, a given parameter in the model. We say that neuron S is *connected* to neuron Ω if $d(N_s(t), \omega) \leq r$.

Observation: this is a directed notion of connectivity as $d(N_s(t), \omega) \leq r \not\Rightarrow d(N_\omega(t), s) \leq r$.

3.1.1 Calculating the distance between axonal trees and soma

In order to use the notion of connectivity, we must be able to calculate this distance for a practical example.

A method for calculating this distance depends on how the neuron is simulated. A fair assumption is to assume that once a neuron is simulated, we have stored the *edge-set* of the axonal tree. This is a set $E \subset \mathbb{R}^2 \times \mathbb{R}^2$ and uniquely defines the tree.

Let S_e represent the segment connecting the two endpoints for an edge $e = [(a_1, b_1), (a_2, b_2)]$. Then we have that,

$$d(N_s(t), \omega) = \min\{d(S_e(t), \omega) : e \in E\} \tag{3.2}$$

where $N_s(t)$ is the axonal tree of a neuron starting at s, E is its edge set and $\omega \in \mathbb{R}^2$.

Thus the distance between an axonal tree and a soma is the minimum distance between the soma and each of the edges of the tree. It still remains to calculate the distance between the soma and an edge.

We follow P. Bourke's method for finding the distance between a point and a segment [4].

Consider a point $\omega = (x, y) \in \mathbb{R}^2$ and an edge-segment S_e connecting (a_1, b_1) and (a_2, b_2). We first calculate u,

$$u = \frac{(x - a_1)(a_2 - a_1) + (y - b_1)(b_2 - b_1)}{||S_e||^2} \tag{3.3}$$

If $u > 1$ then set $u = 1$. If $u < 0$ then set $u = 0$. This is how the method differs from the shortest distance from a point to an entire line.

We can then calculate,

$$\alpha = a_1 + u(a_2 - a_1) \tag{3.4}$$
$$\beta = b_1 + u(b_2 - b_1) \tag{3.5}$$

This gives us the point of intersection of the shortest line, between ω and S_e, and S_e itself. Therefore the shortest distance between S_e and ω is given by,

$$d(S_e, \omega) = ||\omega - (\alpha, \beta)|| \tag{3.6}$$

3.2 Networks of neurons

Following again from Ajazi et al., we assume the soma of the neurons are distributed randomly on some rectangle $[x_1, x_2] \times [y_1, y_2]$ forming a 2D Poisson process with intensity μ [1].

Our model has now expanded its parameter set to include, in addition to the previous parameters,

- $[x_1, x_2], [y_1, y_2]$ - the dimensions of the rectangle in the plane where the soma can be placed.
- r - the radius of connectivity.
- μ - the intensity of the Poisson process governing the number of soma.

3.2.1 Algorithm for a network level model

In order to simulate a network, we first place the soma, by simulating a Poisson process and then grow an axonal from each of these soma using the single neuron model. Finally we determine the connectivity of these neurons and build the adjacency matrix.

In order to relate the theoretical algorithms with the Python implementations, we introduce the following very specific definition.

Definition .3. A *network* is a pair (A, L) where A is an $n \times n$ matrix, called the *adjacency matrix*, and L is a n-dimensional vector of *node-data points*.

The *adjacency matrix*, A, has entries (a_{ij}) given by,

$$a_{ij} = \begin{cases} 1 & \text{if node } i \text{ is connected to node } j \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

A *node-data point* is an ordered list of information about a node. The list has length at least 2, containing the x-position and y-position of the soma. Any additional elements in the list represent hyper-graphical classes that the node is a part of.

We say that the network (A, L) has *order* n.

With this definition, we have a full description of a network to pass between algorithms and to generalise to hyper-graphical structures. Clearly, the position of each node in the plane is also a feature of these models.

We can define the following algorithm for modelling a network.

- Parameters: $\theta_U, \theta_L, \lambda, T, r, \mu, [x_1, x_2], [y_1, y_2]$
  - Area $= (x_2 - x_1)(y_2 - y_1)$
  - Generate n, the number of soma, $n \sim \text{Pois}(\text{Area} \times \mu)$
  - **for** each soma
    - Generate a random x-position, $x \sim \text{U}(x_1, x_2)$
    - Generate a random y-position, $y \sim \text{U}(y_1, y_2)$
    - Add the list $[x, y]$ to L
    - Grow a tree using Algorithm 1 and the given parameters
  - **for** each tree

    - **for** each soma
        - Calculate the distance, d_{ji}, between tree j and soma i
        - **if** $d_{ji} \leq r$
        - Set $a_{ji} = 1$
        - **else**
        - Set $a_{ji} = 0$

- **Output** the network (A, L) where $A = (a_{ij})$

This algorithm outputs the network in a form that is then easy to display using Python code and the `NetworkX` package. There are no hyper-graphical classes for this network and so each node-data point contains only the planar position of the node.

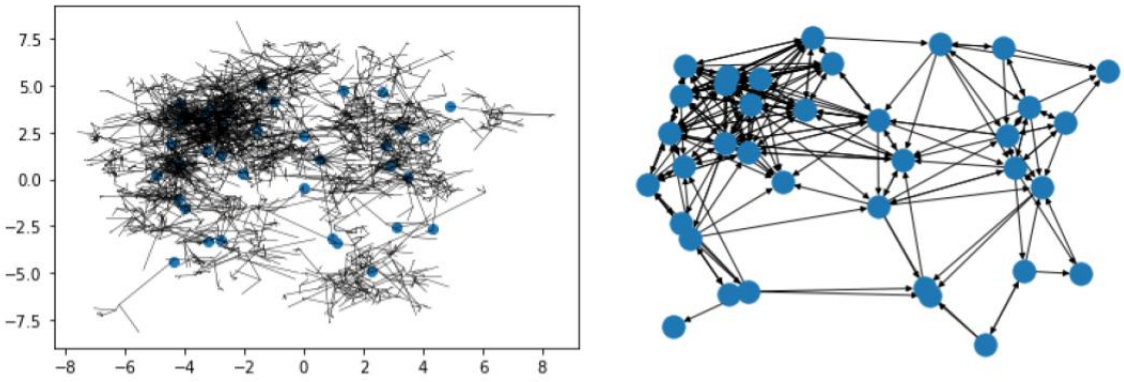

Figure 3: Network level model using parameters: $mu = 0.4, \theta_U = -\theta_L = \pi, T = 4, \lambda = 1, r = 1$

4 Excitatory-inhibitory networks

4.1 Neurotransmitters and Dale's law

Neurotransmitters are chemical messengers that send signals between neurons. If they act in the synaptic cleft between two neurons they can either be *excitatory* or *inhibitory* [5].

Excitatory neurotransmitters promote the generation of action potentials (spikes), whereas inhibitory neurotransmitters prevent them [5].

Dale's law is an important 'rule of thumb' in neuroscience. It states that a neuron releases the same neurotransmitter from all its synapses [6]. This is often taken to mean that a neuron is either excitatory or inhibitory. This gives us a biologically relevant hyper-graphical class to extend the model.

4.2 Excitation & inhibition in computational models

Excitatory-inhibitory (E-I) classifications have been implemented into a range of computational models. Most often, E-I classifications are implemented in spiking and/or recurrent neural networks (SNNs & RNNs) [7]. In these models, one aims to mimic the connectivity and functions of the brain. It is important that excitation does not continue indefinitely in these recurrently connected networks and so inhibitory neurons are implemented.

4.3 Extension to an E-I network model

In order to extend our model to E-I classes, we make some key assumptions. Firstly, we apply Dale's law to say that each neuron is either excitatory or inhibitory for all times and all connections. We make the further assumption that each neuron is classified as either excitatory or inhibitory with some constant probability, given by a random uniform variable. Finally for simplicity, at this stage, we assume that the branching rate for all neurons is the same. In large scale neural networks, it has been observed that this does not hold, but this is investigated in the next chapter on clustered networks [8].

In order to determine the probability of a neuron being either excitatory or inhibitory, we must introduce another parameter. We denote the proportion of inhibitory to excitatory neurons by $\gamma \in [0, 1]$. Experimental results have shown that this varies depending on brain

area. Experimental values vary between 0.07 and 0.48 [8]. A typical value for computational models is 0.2.

Clearly, this will generate a hyper-graphical network with each node-data point now having an additional feature (3 features total): $[x, y, q]$, where q is a symbol that represents the neuron type, $q \in \{\text{Excitatory}, \text{Inhibitory}\}$.

4.3.1 Algorithm for an E-I network model

We can define the following algorithm for modelling an E-I network.

- Parameters: $\theta_U, \theta_L, \lambda, T, r, \mu, [x_1, x_2], [y_1, y_2], \gamma$
  - Area $= (x_2 - x_1)(y_2 - y_1)$
  - Generate n, the number of soma, $n \sim \text{Pois}(\text{Area} \times \mu)$
  - **for** each soma
    - Generate a random x-position, $x \sim \text{U}(x_1, x_2)$
    - Generate a random y-position, $y \sim \text{U}(y_1, y_2)$
    - Generate a random variable $u \sim \text{U}(0, 1)$
      - **if** $u > \gamma$
      - $q = \text{E}$
      - **else**
      - $q = \text{I}$
    - Add the list $[x, y, q]$ to L
    - Grow a tree using Algorithm 1 and the given parameters
  - **for** each tree
    - **for** each soma
      - Calculate the distance, d_{ji}, between tree j and soma i
      - **if** $d_{ji} \leq r$
      - Set $a_{ji} = 1$
      - **else**
      - Set $a_{ji} = 0$
- **Output** the network (A, L) where $A = (a_{ij})$

The simplest way to display this extra classification graphically, is to colour excitatory neurons in black and inhibitory neurons in orange.

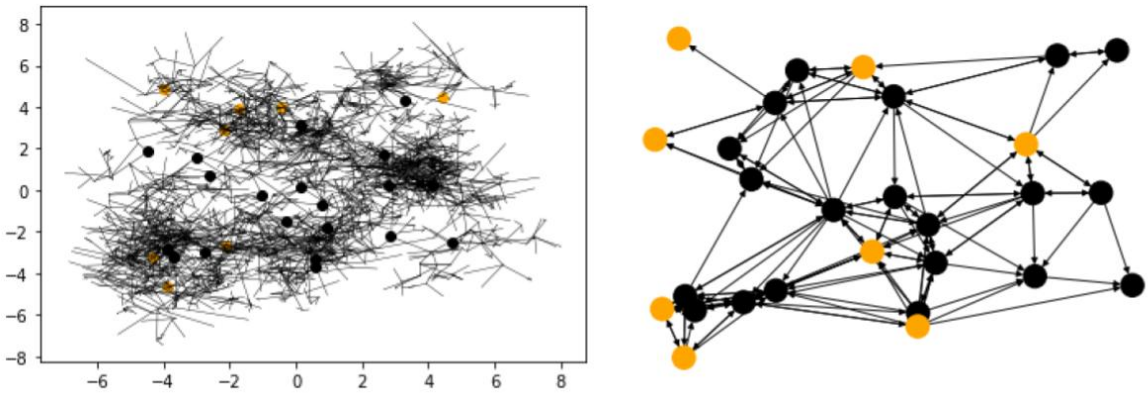

Figure 4: EI Network level model using parameters: $\mu = 0.4, \theta_U = -\theta_L = \pi, T = 4, \lambda = 1, r = 1, \gamma = 0.2$

4.4 Bipartite graphs and the star expansion

There is an equivalence between a bipartite graph and a hyper-graph [9]. If you consider an E-I network, (A, L), it has one hyper-graphical class, neuron type q, which has been represented by colour. We can also perform the *star expansion* to transform our order n hyper-graph into a bipartite graph with $n+2$ nodes. Our two neuron types become nodes in our network and each original node is connected to the node representing its neuron type. As we are dealing with a directed network, we make the further assumption that the original nodes connect **to** the class nodes and that the class nodes are not connected to any other nodes, (i.e they have a 0 out-degree). We can define an algorithm that performs the star expansion on an E-I network.

- Parameters: (A, L) - an E-I network
  - Add two columns and two rows of all zeroes to the A making it a $(n+2) \times (n+2)$ matrix
  - **for** i in 0 to n
    - $[x, y, q] = L[i]$
    - **if** q=Excitatory
      - $A[i, n] = 1$
    - **else**
      - $A[i, n+1] = 1$
  - **Output** A - now a bipartite graph

This concept is presented here because of its relevance to hyper-graphs, but it is not revisited in this project. As seen in Figure

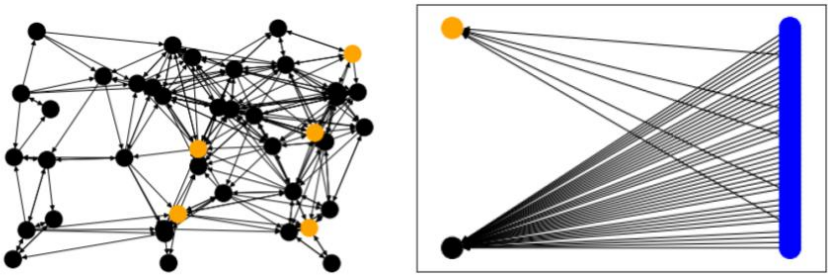

Figure 5: *Left:* E-I network with parameters as in Figure 4. *Right:* Bipartite star graph of the same network generated algorithmically

5, these expanded graphs are less useful visually, as they do not display positions or inter-layer connections very well. For this reason, we represent hyper-graphical classes using different techniques throughout the project.

5 Clustered networks: local and projection neurons

5.1 Organisation in the brain

The human brain is organised into a hierarchical structure of brain areas with different sizes and scales. This scale refers both to mass and volume, but also to the scale of information processing and complexity of represented thought. A large brain area is called a cortex. An example of a cortex is the cerebral cortex which, in turn, is made up of the visual, auditory and somatosensory cortexes [10].

Finding an optimal balance of integration and segregation, with respect to neural connectivity, of brain areas is key to effective processing of information [11]. We can model distinct brain areas in a variety of different ways. One effective way is to consider each brain area as a dense *cluster* of neurons somewhat separated from a distinct cluster. As this is a growth model and therefore a bottom-up model, we do not impose restrictions on what we define to be a cluster, we control the model and the parameters in a motivated way that should produce sufficient clustering and a valid model of distinct brain regions.

5.2 Local and projection neurons

In order to effectively mimic the brain with this clustered model, we must consider a further class of neurons. A simplification of biology, we classify neurons as either *projection* neurons or *local* neurons.

Local neurons typically have short axons and form circuits with nearby neurons whereas projection neurons typically have longer axons and connect different brain areas. This is a valid simplification, because such a classification seems to occur at all scales. For example, local neurons are often referred to as *interneurons*. Within the category of interneurons, are the sub-categories of *local interneurons* and *relay interneurons*. Local interneurons connect to very nearby neurons to form circuits and relay interneurons connect different clusters. This is on a much smaller scale than the local-projection dichotomy mentioned above. This means that whilst the names might change depending on the scale of the brain area being modelled, this neuronal duality occurs in some form [12].

Furthermore, the vast majority of interneurons are inhibitory and

the vast majority of projection neurons are excitatory. This is due to their different functions in information processing [13]. For this reason, we do not consider both classifications simultaneously. However, with simple combination of the two extended models, this would be possible.

5.3 Extension to a clustered network model

5.3.1 Expanding the parameter space

There are many ways one could extend this model to form a clustered network. Here, we will focus on one single method. Firstly we make some key assumptions:

- As the neuron types have physiological differences, we assume that they have a different growth process. This has two effects:
  - The branching rate for the two neurons is different.
  - The Poisson point process dictating the number and position of soma has different intensity.
- We assume further that the radius of connectivity is the same for all neurons.
- The angle bounds are the same for all neurons.
- The branching rate and Poisson process intensity is the same for each cluster. This is a overly strong assumption for modelling some brain areas, however, loosening it increases the parameter space significantly.

By reducing the branching rate, we can cause projection neurons to grow further in a single direction and increase the probability that they reach another cluster. A lower branching rate, for local neurons, will increase the probability that they form connections with nearby neurons.

This means we have to remove two parameters λ, μ and introduce four new ones:

- λ_P - the branching rate for a projection neuron
- λ_L - the branching rate for a local neuron
- μ_P - the intensity of the Poisson point process governing the number of projection neurons

- μ_L - the intensity of the Poisson point process governing the number of local neurons

We assume that $\lambda_L >_P$ in order to model their biological behaviour. The values of μ_P and μ_L vary depending on which brain area you aim to model.

5.3.2 Structuring a clustered network

We consider the rectangle $[x_1, x_2] \times [y_1, y_2]$. Dividing this rectangle in 9 equal sub-rectangles, we label 5 rectangles as the clusters, as shown in Figure 6. We simulate two Poisson point processes in each

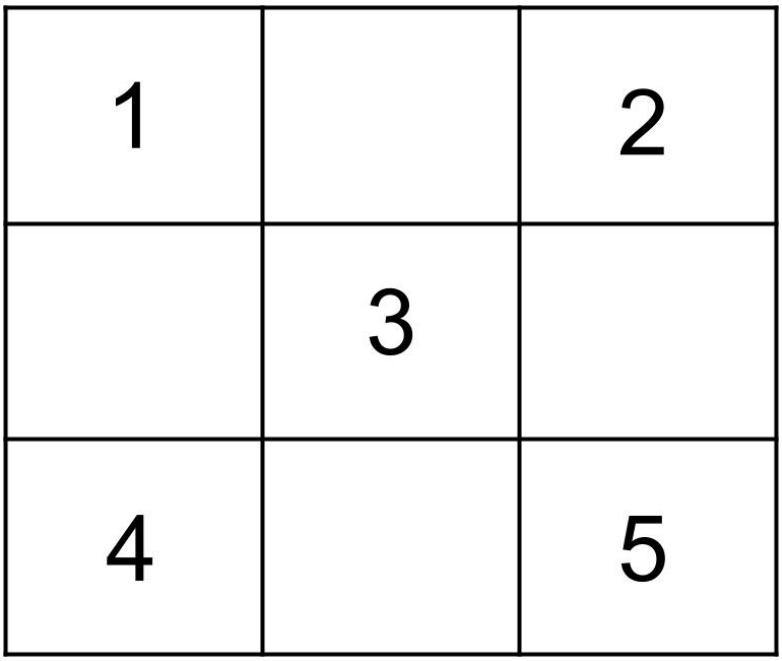

Figure 6: Structure of the clustered network

one of the sub-rectangles, one for each neuron type.

5.3.3 Connectivity of clusters

There is a natural definition of connectivity for clusters.

Definition .4. Cluster A is said to be *connected* to cluster B if $\exists$ n neurons in A s.t. each is connected to a neuron in B.

In other words, a cluster is connected to another if there are at least n connections in that direction. This is a directed notion of

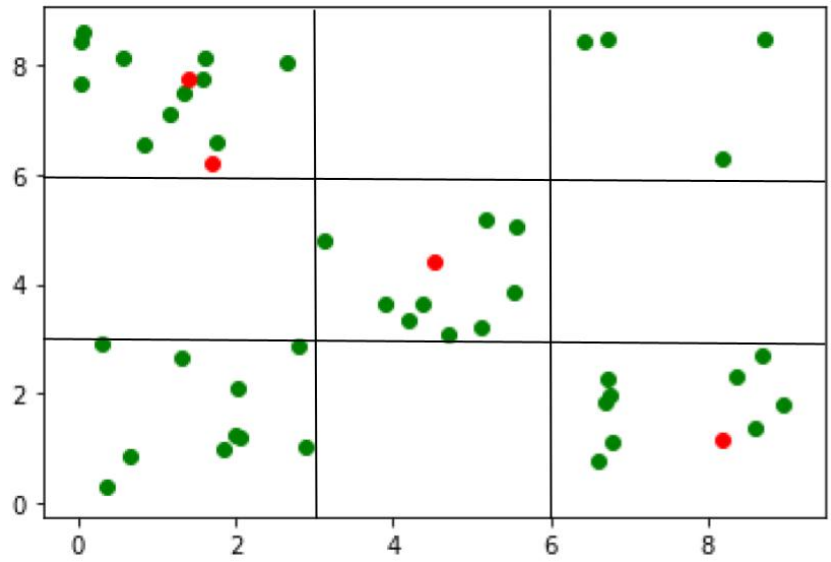

Figure 7: Simulation of clustered soma with $\mu_P = 0.2, \mu_L = 1$. Projection neurons in red and local in green.

connectivity. A valuable extension to this definition of connectivity is to set $n = 1$ and make the connection weighted.

Definition .5. Cluster A is said to be *connected* to cluster B if $\exists$ a neuron in A connected to a neuron in B. The *weight*, w, of this connection is the number of connections between A and B.

This defines a weighted, directed graph with the clusters as nodes. This can be an important tool to analyse the level of connectivity between clusters.

The clustered network (A, L) is clearly a hyper-graphical structure with multiple classes. Each node-data point in L now has four items:

- x-position $\in [x_1, x_2]$
- y-position $\in [y_1, y_2]$
- k - neuron type $\in$ {Local, Projection}
- c - cluster $\in \{1, 2, 3, 4, 5\}$

As mentioned earlier, the vast majority of local neurons are inhibitory and the vast majority of projection neurons are excitatory, so we consider this class redundant.

In order to display the various classes, we do not need to go to

higher level structures. We can display the type of neuron with colours: green for local and red for projection. The cluster is obvious from the position of the neuron in the network (it is displayed in the (x, y) position of the soma). Therefore, all the hierarchical information can be encoded into a simple, coloured, 2D network.

5.3.4 Algorithm for a clustered network model

We can define the following algorithm for modelling a clustered network.

- Parameters: $\theta_U, \theta_L, \lambda_P, \lambda_L, T, r, \mu_P, \mu_L, [x_1, x_2], [y_1, y_2]$
  - **for** each cluster
    - Area $= (x_2^c - x_1^c)(y_2^c - y_1^c)$
    - Generate n_L, the number of local soma, $n \sim$ Pois(Area$\times$ μ_L)
    - Generate n_P, the number of projection soma, $n \sim$ Pois(Area $\times$ μ_P)
    - **for** each soma
      - Generate a random x-position, $x \sim \mathrm{U}(x_1^c, x_2^c)$
      - Generate a random y-position, $y \sim \mathrm{U}(y_1^c, y_2^c)$
      - Add the list $[x, y, k, c]$ to L (k- neuron type, c - cluster)
      - **if** k=Local
      - Grow a tree using Algorithm 1 and the given parameters with $\lambda = \lambda_L$
      - **else**
      - Grow a tree using Algorithm 1 and the given parameters with $\lambda = \lambda_P$
  - **for** each tree
    - **for** each soma
      - Calculate the distance, d_{ji}, between tree j and soma i
      - **if** $d_{ji} \leq r$
      - Set $a_{ji} = 1$
      - **else**
      - Set $a_{ji} = 0$
- **Output** the network (A, L) where $A = (a_{ij})$

Note: here $[x_1^c, x_2^c] \times [y_1^c, y_2^c]$ is the region bounding cluster c. This is easy to workout using the structural diagram, but the formulae are given in the appendix A.

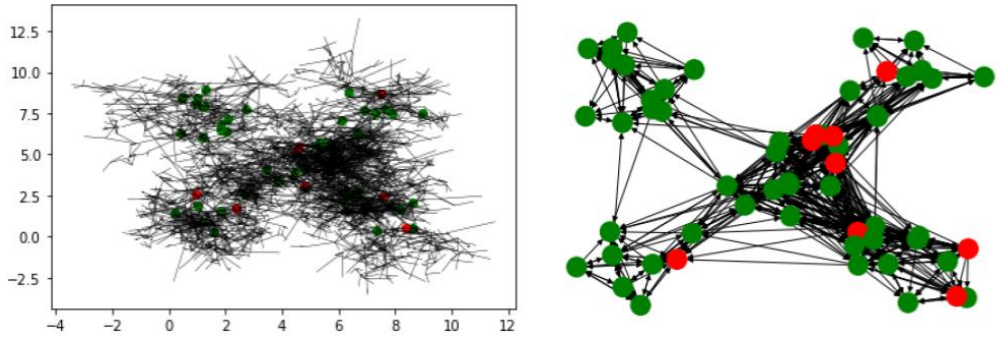

Figure 8: Clustered network model: $\mu_L = 1, \mu_P = 0.2, [x_1, x_2] \times [y_1, y_2] = [0, 9] \times [0, 9], \theta_P = -\theta_L = \pi, T = 4, \lambda_L = 1, \lambda_P = 0.5, r = 1$

5.3.5 Algorithm for a weighted cluster model

As mentioned before, from this network, we can derive a directed, weighted graph with clusters as nodes and weighted edges representing the number of connections between a cluster. Again, to help the theory line up with the implementation, we introduce a specific definition.

Definition .6. A *weighted, directed, cluster graph* is an $n \times n$ matrix M with entries,

$$m_{ij} = n \tag{5.1}$$

where n is the number of connections between cluster i and cluster j.

Observation: as the graph is directed the matrix may not be symmetric.

We can define the following algorithm to go from a clustered network model to a weighted, directed, cluster graph.

- Parameters: (A, L) - clustered network of order n
- **for** each cluster c_1
  - **for** each cluster c_2
  - Count the cluster connections between c_1 and c_2 by:
  - Connections initialised as $\omega = 0$
    - **for** i in 0 to n
    - $[x, y, k, c] = L[i]$
    - if $c = c_1$

    - **for** j in 0 to n
    - $[x^*, y^*, k^*, c^*] = L[j]$
    - **if** $A[i, j] = 1$ and $c^* = c_2$
    - $\omega = \omega + 1$
  - $m_{c_1,c_2} = \omega$
- **Output** $M = (m_{ij})$

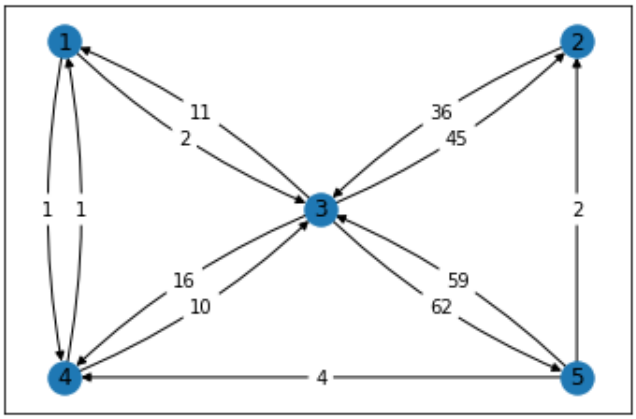

Figure 9: Weighted cluster graph of cluster network in Figure 8

6 Machine learning: layered neural networks

6.1 Neuromorphic machine learning

At the intersection of computational neuroscience and machine learning is the field of *neuromorphic machine learning (NML)*. Researchers in this area aim to build biologically motivated/plausible artificial intelligence (AI) systems, often with the end goal of whole brain emulation and conscious AI.

One long standing limitation with machine learning neural networks, which we will refer to as layered neural networks, is that they act as *black boxes* with little to no mathematical theory explaining why one topology/connectivity works better than another [14].

Whilst traditional machine learning disciplines may be interested in solving the above problem, NML researchers aim to mimic the connectivity in the brain whilst still having a network capable of learning.

6.2 Extension to a layered neural network model

In order to extend this model to the simulation of layered neural networks, there are some key differences that need to be factored in to the design.

Firstly, model training algorithms in ML often use *weight matrices* to represent the weight of a connection between two neurons. These $n \times n$ matrices have dimensions that assume full connectivity between layers. However, this is not a limitation as we can represent a lack of connection with a 0 in the weight matrix.

Furthermore, there is less freedom with choice over number of neurons in the input/output layer of a layered neural network. The number of input and output neurons is often specified by the nature of the problem and therefore cannot be modelled by some random process. For example, the classical introductory machine learning problem, the recognition of handwritten characters from the NMIST data set, maps each character to a 28×28 pixel image [15]. This specifies that there must be 784 input neurons. Typically, for a classification problem like this, there is an output neuron for each potential classification, i.e. one for each digit 0-9. This specifies 10 output neurons.

In order for this model to be useful for NML, the number of input and output neurons should be specified as parameters to the model. Furthermore, another feature depending on the nature of the problem, and the nature of the research, is the number of hidden layers. This should also be specified as a parameter to the model. We do not specify the number of neurons in each hidden layer, as an optimum value is not obvious from the problem and is better simulated from the model. This gives the model three new parameters:

- N_I - the number of neurons in the input layer.
- N_O - the number of neurons in the output layer.
- H - the number of layers.

We are also assuming here that the intensity of the Poisson process that controls the number of neurons in a layer is constant for all hidden layers.

In the layered neural network, as it is being used for learning, position in $\mathbb{R}^2$ is of far reduced importance. The y-position is useful for sorting position in a layer, but $x-$position becomes redundant as it gets factored into the hyper-graphical class of 'layer'. As such, a node data point for a layered neural network is made up of only two classes:

- l - layer.
- p - position in layer (from the top).

This gives us the restriction $1 \leq l \leq H$.

6.2.1 Structuring a layered neural network

In this model extensions, we discard the parameter $[x_1, x_2] \times [y_1, y_2]$ as space is of reduced importance. We now restrict the y-position of any generated soma to be in the range $[0, 8]$. The x-position must be in the range $[0, 2H]$. If the x-position of a soma is in the range $[2(l-1), 2l]$, we say that it is in the layer l. We now have the method for simulating a layered neural network. First, we explain in high level steps,

1. Distribute N_I soma in the strip $[0, 2] \times [0, 8]$.
2. Distribute N_O soma in the strip $[2(H-1), 2H] \times [0, 8]$.

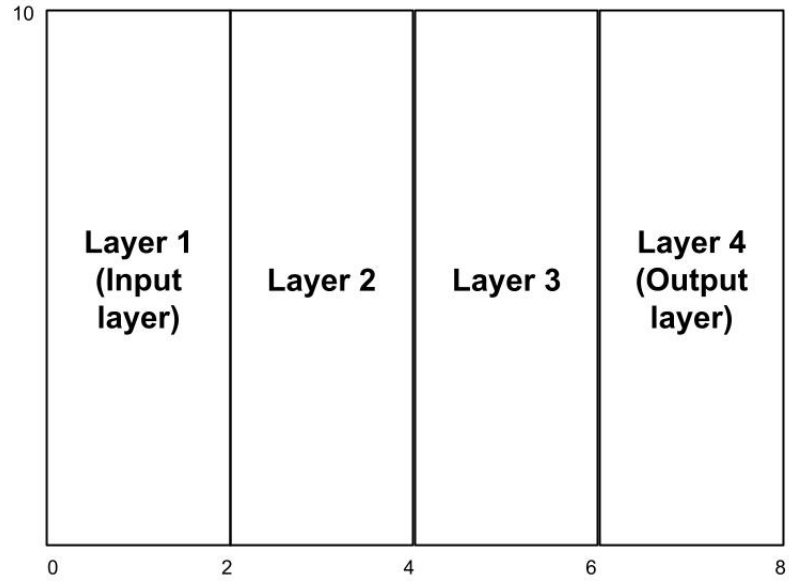

Figure 10: Structure of a layered neural network with four layers.

3. For each hidden layer, generate the number of soma using the Poisson point process with area 16 and distribute in the strip.
4. For each soma, grow a tree with narrow angle bounds and record the connectivity of the network, only allowing connection between layer l and $l + 1$.
5. Use the y-position of the soma to order the neurons in their layers and discard the x-position.
6. Display the result as a directed network.

6.2.2 Algorithm for a layered neural network

We can now define the algorithm for simulating a layered neural network.

- Parameters: $\mu, N_I, N_O, H, \theta_L, \theta_U, T, \lambda, r$
- **for** i in 0 to N_I
- Generate a random (x, y) position. $(x, y) \sim (\mathrm{U}(0, 2), \mathrm{U}(0, 8))$
- Add node data point $[x, y, 1]$ to L
- **for** i in 0 to N_O

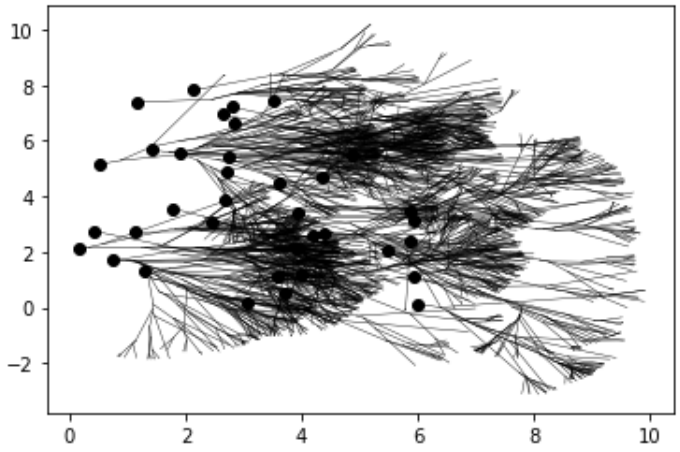

Figure 11: Grown layered neural network: $\mu = 1, N_I = 10, N_O = 10, H = 3, \theta_U = -\theta_L = \frac{\pi}{6}, T = 4, \lambda = 1, r = 1$

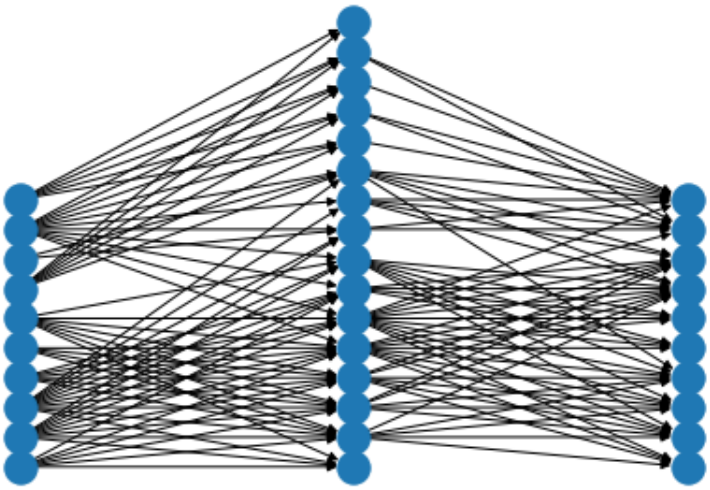

Figure 12: Layered neural network displaying the network grown in Figure 11.

- Generate a random (x, y) position. $(x, y) \sim (\mathrm{U}(2(H-1), 2H), \mathrm{U}(0, 8))$
- Add node data point $[x, y, H]$ to L
- **for** j in 0 to $H - 2$
- Generate $N_{j+2} \sim \mathrm{Pois}(16\mu)$. This is the number of soma in layer $j + 2$
- Generate a random (x, y) position. $(x, y) \sim (\mathrm{U}(2(j + 1), 2(j + 2)), \mathrm{U}(0, 8))$

- Add node data point $[x, y, j + 2]$ to L
- **for** each soma
  - Grow a random tree using the parameters
- **for** each soma i
  - **for** each tree j
    - Calculate d_{ji}, the distance between the tree j and the soma i
    - $[x, y, l] = L[i]$
    - $[x^*, y^*, l^*] = L[j]$
    - **if** $d_{ji} \leq r$ **and** $l^* + 1 = l$
    - Set $a_{ji} = 1$
    - **else**
    - Set $a_{ji} = 0$
- Let K be a H- dimensional list of lists
- **for** each node-data point $[x, y, l] \in L$
  - Add $[x, y, l]$ to $K[l - 1]$. This sorts nodes into layers.
- **for** each layer k in K
  - Sort by y and let p be the new (discrete) position of each soma in the layer
  - **for each** node in the layer k
    - Add $[l, p]$ (the new node-data point) to L^*
- **Output** (A, L^*) where $A = (a_{ij})$

Whilst this network has two hyper-graphical classes, both can be represented by the position of the node; by which layer it is in and by its position in that layer. No further hyper-graphical representation is needed.

6.2.3 Filtering unused nodes

The model has the ability to produce some networks with connections that do not influence the outcome of learning in the network. For example,

1. Input nodes may not connect to the subsequent layer meaning that an input remains unused.
2. A node may have no connections and remain unused.

3. A node may have output connections but no input connections.
4. A node may have input connections but no output connections.
5. A node in the output layer may have no connections, making that classification impossible to achieve.

Each one of these cases requires a different solution:

1. Guaranteeing connections compromises the underlying principles of the model. In this case, one can either use the network - this is fine if N_I is very high - or simulate another - this is better if N_I is low.
2. We seek to remove unused nodes which do not add to the network. We do **not** remove input or output nodes even if they are unused.
3. These nodes can be important in probabilistic machine learning or machine learning with stochastic activation functions as they can still add noise. This is actually beneficial for sampling networks that aim to produce independent samples of a posterior distribution from data. If necessary, the input can be assumed to be 0. As such, we leave them in.
4. We seek to remove these nodes which do not add to the network.
5. Again, guaranteeing connections compromises the underlying principles of the model. If the network is being used for a regression problem, it can still be used. If it is being used for a classification problem, some classifications are impossible to achieve and so another network should be simulated.

In simpler terms, to be removed, a node must be in a hidden layer and have no outward connections.

6.3 E-I layered neural networks

In the field of NML, particularly with spiking neural networks, E-I classes are considered in learning networks. By combing the ideas from Chapter 4, we can simulate an E-I layered neural network. We reintroduce the γ parameter (proportion of neurons that are inhibitory). Whilst the classification of output neurons has no effect on the learning algorithm, we still classify them as either inhibitory or excitatory for consistency and biological plausibility. Each node data point would now have three features:

- l - layer.

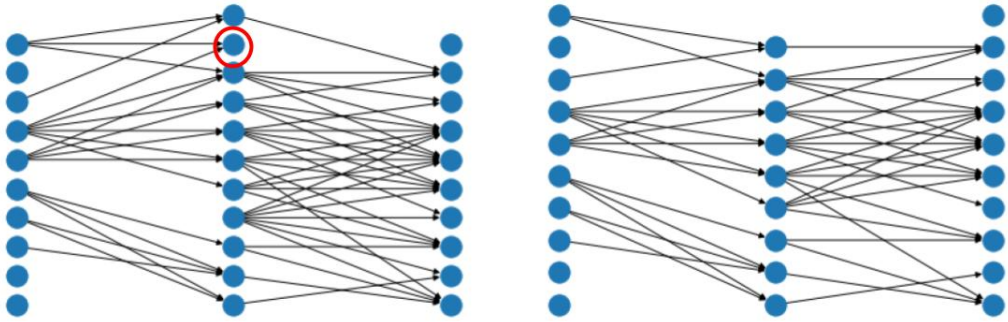

Figure 13: *Left:* A layered neural network simulated with parameters as in Figure 12 but with $T = 2$. *Right* the same network with the unused node removed algorithmically.

- p - position in layer.
- q - neuron type $\in \{\text{Excitatory}, \text{Inhibitory}\}$

The algorithm remains almost unchanged except we add one stage where we classify each node into either excitatory or inhibitory in the same way as chapter 4.

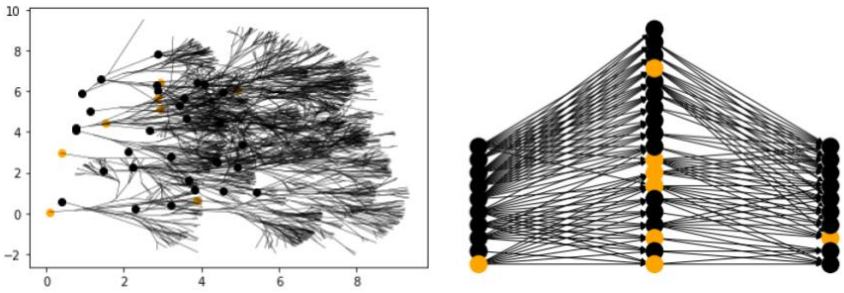

Figure 14: Grown E-I layered neural network: $\mu = 1, N_I = 10, N_O = 10, H = 3, \theta_U = -\theta_L = \frac{\pi}{6}, T = 4, \lambda = 1, r = 1$

7 Hopfield networks

7.1 Features of a Hopfield network

Hopfield networks are an example of a recurrent neural network. This means that nodes can connect into themselves, either directly or through a path. Hopfield network are also an example of a network used for learning, with the particularity that they have 'memory'. They are not organised into layers, and as such have no 'output'. The output of a Hopfield network is the entire state of the network at a given time. For this reason they linked heavily to the study of dynamical systems [16].

Hopfield networks are often modelled as fully connected, recurrent networks. However, full connectivity is not a requirement as lack of connection can be represented as 0 in the weight matrix. The networks also have symmetric connections with symmetric weights. The positions of nodes in space are not considered in the Hopfield network.

7.2 Extension to a Hopfield network model

The Hopfield network is the simplest model in this project. It is not a hyper-graphical structure and has no additional classes; even the (x, y)-position of the soma is redundant. We can therefore narrow the definition of a Hopfield network significantly from the more general network.

Definition .7. A *Hopfield network*, H, is a symmetric matrix, $H = H^\top$, where the entries are:

$$h_{ij} = \begin{cases} 1 & \text{if node } i \text{ and node } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

We do not consider the weights in this definition as we are focused on generating a Hopfield structure rather than a learning algorithm.

We also need to modify our notion of connectivity for the Hopfield network so that it is symmetric.

Definition .8. Let A, B be two neurons simulated with the algorithm defined in section 3.2.1. We say that A, B are *connected in the Hopfield network* if either A is connected to B or B is connected to A (using the definition of connected in Definition .2).

7.2.1 Algorithm for a Hopfield network model

Using this definition of connectivity, we can define the following algorithm for simulating a Hopfield network.

- Parameters: $\theta_U, \theta_L, \lambda, T, r, \mu, [x_1, x_2], [y_1, y_2]$
  - Area $= (x_2 - x_1)(y_2 - y_1)$
  - Generate n, the number of soma, $n \sim \text{Pois}(\text{Area} \times \mu)$
  - **for** each soma
    - Generate a random x-position, $x \sim \text{U}(x_1, x_2)$
    - Generate a random y-position, $y \sim \text{U}(y_1, y_2)$
    - Add the list $[x, y]$ to L
    - Grow a tree using Algorithm 1 and the given parameters
  - Let H be an $n \times n$ matrix with 0 entries
  - **for** each tree
    - **for** each soma
      - Calculate the distance, d_{ji}, between tree j and soma i
      - **if** $d_{ji} \leq r$
      - Set $h_{ji} = 1$
      - Set $h_{ij} = 1$
- **Output** the Hopfield network H

We display the Hopfield network as a circular graph as the (x, y)-positions of the soma are not relevant (nor outputted).

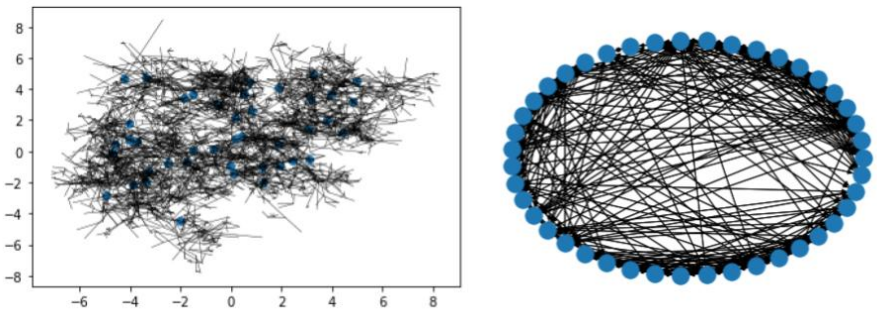

Figure 15: Grown Hopfield network: $mu = 0.4, \theta_U = -\theta_L = \pi, T = 4, \lambda = 1, r = 1$

7.3 E-I Hopfield network

Hopfield networks are closely linked to biologically inspired learning algorithms. As such the treatment of Hopfield network can be extended to E-I Hopfield networks. Following on from chapter 4, we can introduce an E-I classification into our Hopfield algorithm without adding much complexity. Again, this yields a hyper-graphical structure that we represent with colours for the two classes.

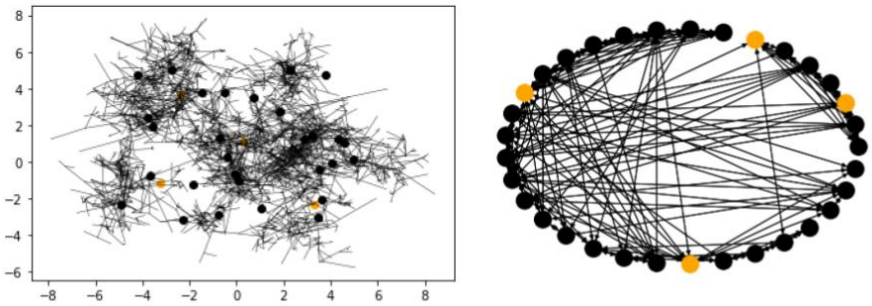

Figure 16: E-I grown Hopfield network: $mu = 0.4, \theta_U = -\theta_L = \pi, T = 4, \lambda = 1, r = 1, \gamma = 0.2$

References

[1] F. Ajazi, G. M. Napolitano, T. Turova, and I. Zaurbek, "Structure of a randomly grown 2-d network," *Biosystems*, vol. 136, pp. 105–112, 2015. Selected papers presented at the Eleventh International Workshop on Neural Coding, Versailles, France, 2014.

[2] T. B. V. H. P. P. F. D. R. A. R. G. J. A. V. P. J. Koene, R. A. and A. Van Ooyen, "Netmorph: A framework for the stochastic generation of large scale neuronal networks with realistic neuron morphologies," *Neuroinformatics*, vol. 7, p. 195– 210, 2009.

[3] M.-M. T. H. R. T. H. L. M.-L. Acimovíc, J., "Modeling of neuronal growth in vitro: comparison of simulation tools netmorph and cx3d," *Bioinformatics*, 2011.

[4] P. Bourke, "Minimum distance between a point and a line," 1988.

[5] Q. B. Institute, "What are neurotransmitters?," 2018.

[6] H. Dale, "Pharmacology and nerve-endings," 1934.

[7] X.-J. W. H. Francis Song, Guangyu R. Yang, "Training excitatory-inhibitory recurrent neural networks for cognitive tasks: A simple and flexible framework," *PLos Computational Biology*, 2016.

[8] Z. E. Attila I. Gulyás, Manuel Megías and T. F. Freund, "Total number and ratio of excitatory and inhibitory synapses converging onto single interneurons of different types in the ca1 area of the rat hippocampus," *Journal of Neuroscience*, 1999.

[9] M. Kirkilionis, "Structures of complex systems," 2021.

[10] S. Shipp, "Structure and function of the cerebral cortex," *Current Biology*, 2007.

[11] T. J. A. . C. G. Abdallah, "Determining the hierarchical architecture of the human brain using subject-level clustering of functional networks," *Scientific Reports*, 2019.

[12] J. J. T. Kandel, Eric; Schwartz, *Principles of Neural Science*. 2000.

[13] T. I. Fujii H, "Interneurons: their cognitive roles-a perspective from dynamical systems view.," 2005.

[14] G. Self, "Randomly wired neural networks and state-of-the-art accuracy? yes it works.," *towards data science*, 2019.

[15] S. Patel, "A-z handwritten alphabets in .csv format," 2017.

[16] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc Natl Acad Sci U S A*, 1982.

A Region bounding a cluster

As mentioned in chapter 5, here we give the formulae for the regions bounding the five clusters. Consider the rectangle $[x_1, x_2] \times [y_1, y_2]$, and let $[x_1^c, x_2^c] \times [y_1^c, y_2^c]$ be the region bounding cluster c. First we define

$$\Delta x = \frac{x_2 - x_1}{3} \tag{A.1}$$

$$\Delta y = \frac{y_2 - y_1}{3} \tag{A.2}$$

Then, following on from Figure 6, we have,

$$[x_1^1, x_2^1] \times [y_1^1, y_2^1] = [x_1, x_1 + \Delta x] \times [y_2 - \Delta y, y_2] \tag{A.3}$$

$$[x_1^2, x_2^2] \times [y_1^2, y_2^2] = [x_2 - \Delta x, x_2] \times [y_2 - \Delta y, y_2] \tag{A.4}$$

$$[x_1^3, x_2^3] \times [y_1^3, y_2^3] = [x_1 + \Delta x, x_2 - \Delta x] \times [y_1 + \Delta y, y_2 - \Delta y] \tag{A.5}$$

$$[x_1^4, x_2^4] \times [y_1^4, y_2^4] = [x_1, x_1 + \Delta x] \times [y_1, y_1 + \Delta y] \tag{A.6}$$

$$[x_1^5, x_2^5] \times [y_1^5, y_2^5] = [x_2 - \Delta x, x_2] \times [y_1, y_1 + \Delta y] \tag{A.7}$$