LAM Autopsy

University of Sheffield
PSY6431 | 220225292

INTRODUCTION

Multiscale representations of community structures in attractor neural networks (Haga and Fukai, 2021) demonstrates that information representations in the attractor state are related to graph Laplacian eigenvectors, a popular mathematical technique for graph segmentation and non-linear dimensionality reduction.

"Our model establishes a theoretical relationship between associative memory networks and graph theory, providing a biologically plausible dynamical mechanism for hierarchical abstraction in the brain." (Haga and Fukai, 2021)

The authors indicate, the scale of representations (i.e. size of the represented communities) is modulated by the relative strength of local and global inhibitory circuits, an active role of target-specific and inhomogeneous neuromodulation. This autopsy examines an implementation of the paper (*in silico*) to explore possible exploits for future research.

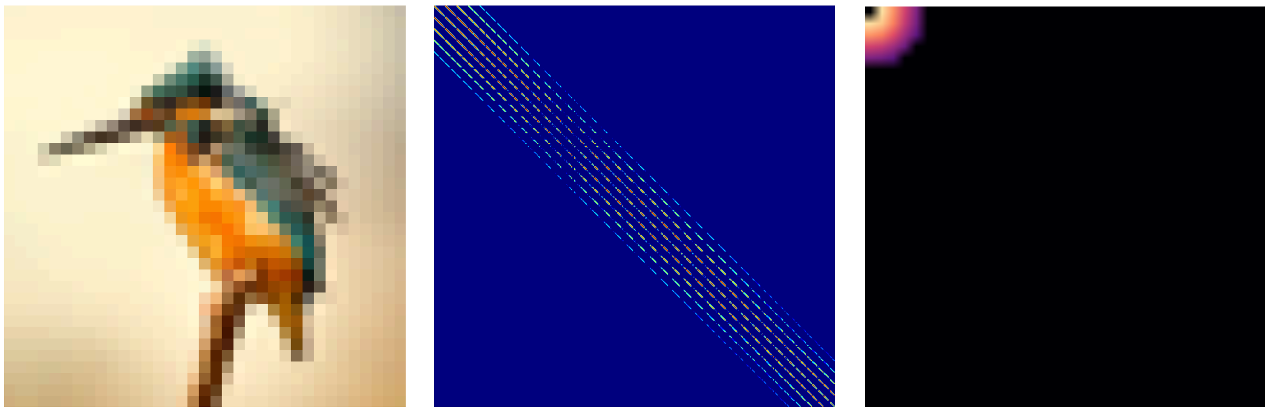
CONSTRUCT GRAPH

To study the higher dimensional community relationships, both the laplacian eigenvectors and the neural network receive a normalised adjacency matrix constructed from input stimulus (*source image*). This matrix is defined by the spatial and luminance properties between two corresponding pixels. By constructing two opposing lists of every location on the input, the euclidean distance between each set of vectors can then be calculated and if they are within a defined search radius *r*, then appended to a new list to be processed under the following conditions:

Luminance | The sum of the squared Euclidean distance between two pixels values across all channels divided by width parameter *sigmaI* which defines the width of the Gaussian kernel. Using the co-ordinates to index the stimulus, the result is negated and exponentiated to ensure similar values between pixels are weighted more heavily than dissimilar.

Spatial | Computing the spatial proximity between two vectors, the term exponentiates the negative of this distance to create a gradient representing neighbouring pixels with nearby domains have a stronger weight than distant pixels, again using *sigmaX* to define the with of the spatial kernel.

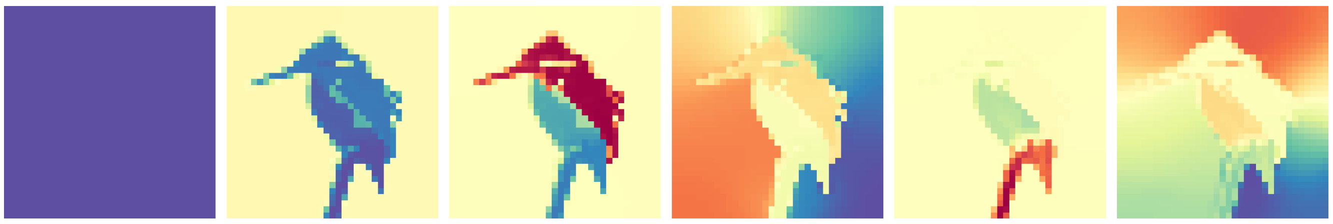
These terms are multiplied together to provide a *simval* for each pixel in reference to its surrounding neighbours located within *r*, providing the final adjacency matrix.



1 | Resized kingfisher source image. 2 | Adjacency matrix. 3 | Sequential render of neighbourhood weight relationship, where each frame represents a row in the adjacency matrix

EIGENVECTORS

To compare information is representations in attractor neural network correspond to Laplacian eigenvector, the adjacency matrix is normalised either as a *symmetric* or *asymmetric* degree matrix *norm_mode;* the Eigen -values and -vectors are calculated and ordered accordingly. *Note: H or L = Dnorm* * A



Principle component analysis of the representative GL eigenvectors from a normalised asymmetric adjacency matrix.

NORMALISATION

The model presents two methods of normalisation, *symmetric* and *asymmetric*. Both approaches create a diagonal matrix where each element is the sum of the row matrix relating to the number of edges attached at each vertex. The difference between each approach is the exponent value and the multiplication of the hetero-associative weights.

Symmetric normalisation | $\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$

Enabling formal theoretical analysis. The hetero-associative weights are defined by performing left & right -multiplication to ensure both rows and columns are normalised and symmetric, this double normalisation also defines the power of 0.5 dividing each column of the result by the same value. When defining the local inhibition, excitatory and global inhibitory weights, symmetric mode uses the bias of the generated pattern,

Asymmetric normalisation | $\cdot\, \mathbf{D}^{-1}\mathbf{A}$

As with the symmetric approach, asymmetric creates a diagonal matrix from the sum of edges adjacent for each node normalised to a value of 1. The matrix multiplication is performed only where left-multiplies *Dnorm,* the degree matrix with the original hetero-associative weights, finally replacing the original self.H matrix; transforming the adjacency matrix into a Laplacian matrix. This single multiplication allows asymmetry with only the rows are normalised. For the local inhibition, excitatory and global inhibitory weights, instead of using the bias the binary dipole pattern is used.

Hetero-associative weights |

When a new pattern is presented to the network, the weights retrieve the stored pattern most similar to the input pattern. *Note: Unlike auto-associative which recover or complete missing or corrupts parts of the signal.*

INITIALISATION

self.N | Number of neurons, default = 30000
self.P | Number of random memory patterns, 1 for each pixel in stimulus
self.prob | Sparsity, probability of a pixel being 0-1 in the generated pattern
self.H | Hetero-associative weights, adjacency matrix input
self.gamma | Strength of global inhibition, a higher value = stronger inhibition promoting more competition among neurons.
self.norm_mode | Choice between symmetric (sym) and asymmetric (asym) normalisation.
self.V | Negated probability = self.prob * (1-self.prob), used in RL (simulate_allstarts)
self.NV | Number of neurons divided by negated probability, normalisation factor.

self.xi | A binary matrix [self.N, self.P]; returning a either 0-1, randomly determined by the probability outcome of self.prob.
self.xi_mean | Sum of rows self.xi returns the activation of each neuron for all input patterns, divided by self.P for mean activation.
self.xi_bias | Subtracting the mean activation away from self.xi entering the activation of each neuron around zero.

```
self.Wauto = (self.xi @ self.xi.T) / self.NV
```

Local inhibition of the network, taking the outer product of *self.xi* with itself, then dividing by the number of input vectors *self.NV* to adjust the strength of local inhibition. AKA, the auto-correlation weight matrix, which captures the correlations between the neurons in the absence of any external input.

```
self.Whetero = (self.xi @ self.H @ self.xi.T)
```

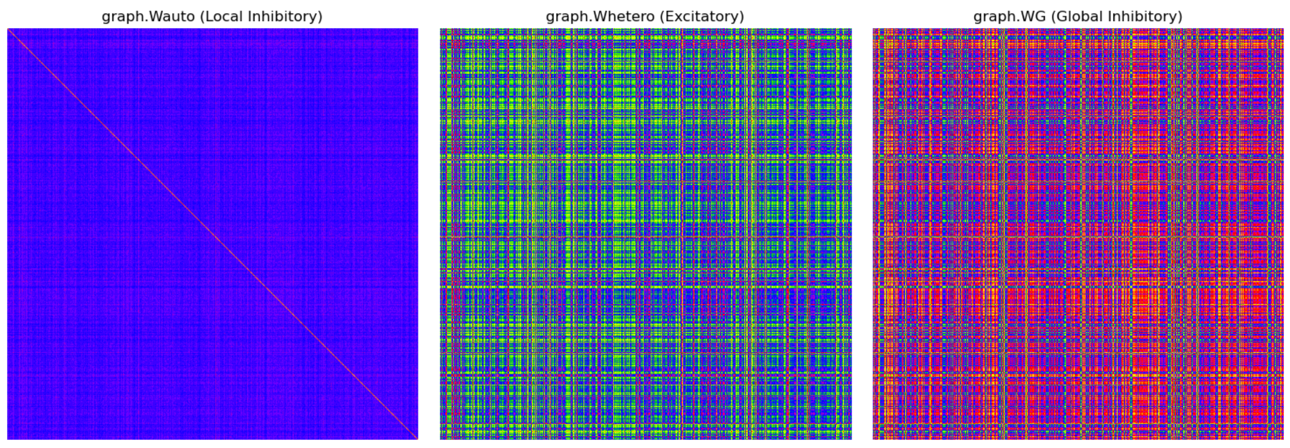
Excitatory connections between neurons. Constructed by calculating the outer-product of the input pattern self.xi and self.H, again using self.NV to control the strength of excitatory connections. AKA the hetero-correlation weight matrix, capturing the correlations between the neurons induced by the external input through the connectivity matrix *self.H.*

```
self.WG = self.P * self.xi_mean @ self.xi_mean.T / self.NV + self.gamma / self.N
```

Global inhibition of the network, taking the outer product of self.xi_mean with itself, then scaled by a factor *self.P*. *self.WG* provides a global inhibitory input to the neurons, which counteracts any excitatory input ensuring the neuron is bounded and does not diverge.

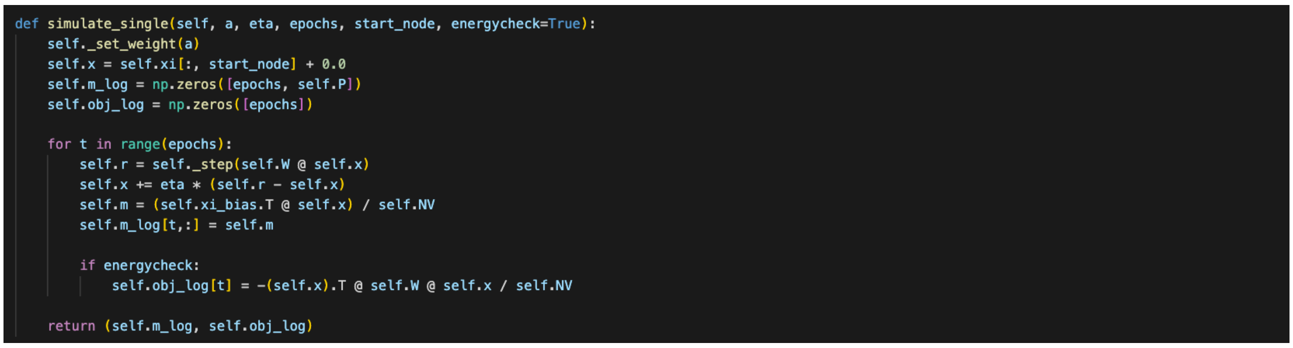
```
self.W = a * self.Wauto + self.Whetero - (a+1) * self.WG
```

Alpha *a* is used to define the weight of *self.Wauto* relative to *self.Whetero*, then subtracting *a*+1 times the global inhibition to return *self.W.* A higher *a* favours auto-correlation, modelling a neuron being active when previous activity was high, while a lower *a* value favours hetero-correlation modelling the influence of activity on other neurons.



Visual representation of all the connectivity matrices used to construct the graph weights.

COMPUTE GRAPH



```python
def simulate_single(self, a, eta, epochs, start_node, energycheck=True):
    self._set_weight(a)
    self.x = self.xi[:, start_node] + 0.0
    self.m_log = np.zeros([epochs, self.P])
    self.obj_log = np.zeros([epochs])

    for t in range(epochs):
        self.r = self._step(self.W @ self.x)
        self.x += eta * (self.r - self.x)
        self.m = (self.xi_bias.T @ self.x) / self.NV
        self.m_log[t,:] = self.m

        if energycheck:
            self.obj_log[t] = -(self.x).T @ self.W @ self.x / self.NV

    return (self.m_log, self.obj_log)
```

Coded function to simulate a single iteration of the network depending on a defined alpha value.

self.r | Response, dot product of weight matrix (self.W) and current network state (self.x) followed by a thresholding operation which performs a threshold activation where Θ(x) is a step function (Θ(x) = 1 if x>0, otherwise Θ(x) = 0.
self.x | Network update (GD), updating neuron activity as a weighted (eta) average of previous activity (x) to current input (r).
self.m | Pattern overlap, a measure of similarity between the state of the neuron and the average state of its neighbours.

The function initialises the network's weights using alpha to define the relative weights between Wauto, Whetero and WG. The initial condition is defined using a start_node which is the centre cell index of the number of patterns (35x35 / 2 = 1225/2 = 612). Self.xi is a binary matrix that specifies the state of each node, setting the init_condition of each node to match the state of this indexed node.

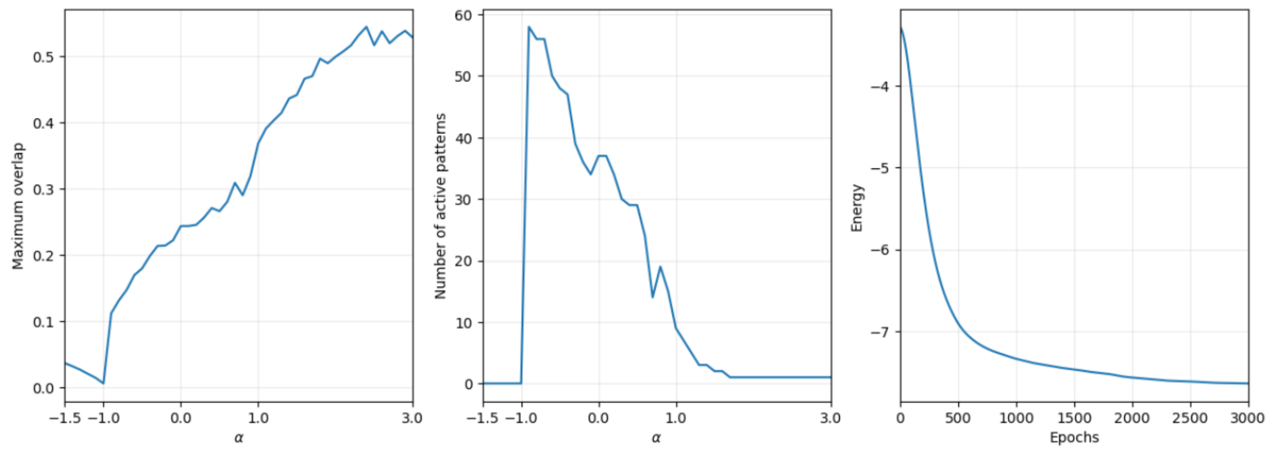
The method then runs a loop over a specified number of epochs (steps) and at each time step:

1. Computes the input to the neuron as a dot-product of the weight matrix and the current state of *self.x.*
2. The result of the matrix multiplication is passed through a heaviside step function Θ, where, (Θ(x) = 1 if x>0, else Θ(x) = 0.
3. Update the state of the network using gradient descent based on the difference between the thresholded input *self.r* and the current network *self.x*, where the speed of convergence is controlled by a learning rate *eta*.
4. *self.m* is the mean activity of each pattern, evolving over each step of the simulation improves as the network falls into a steady state.

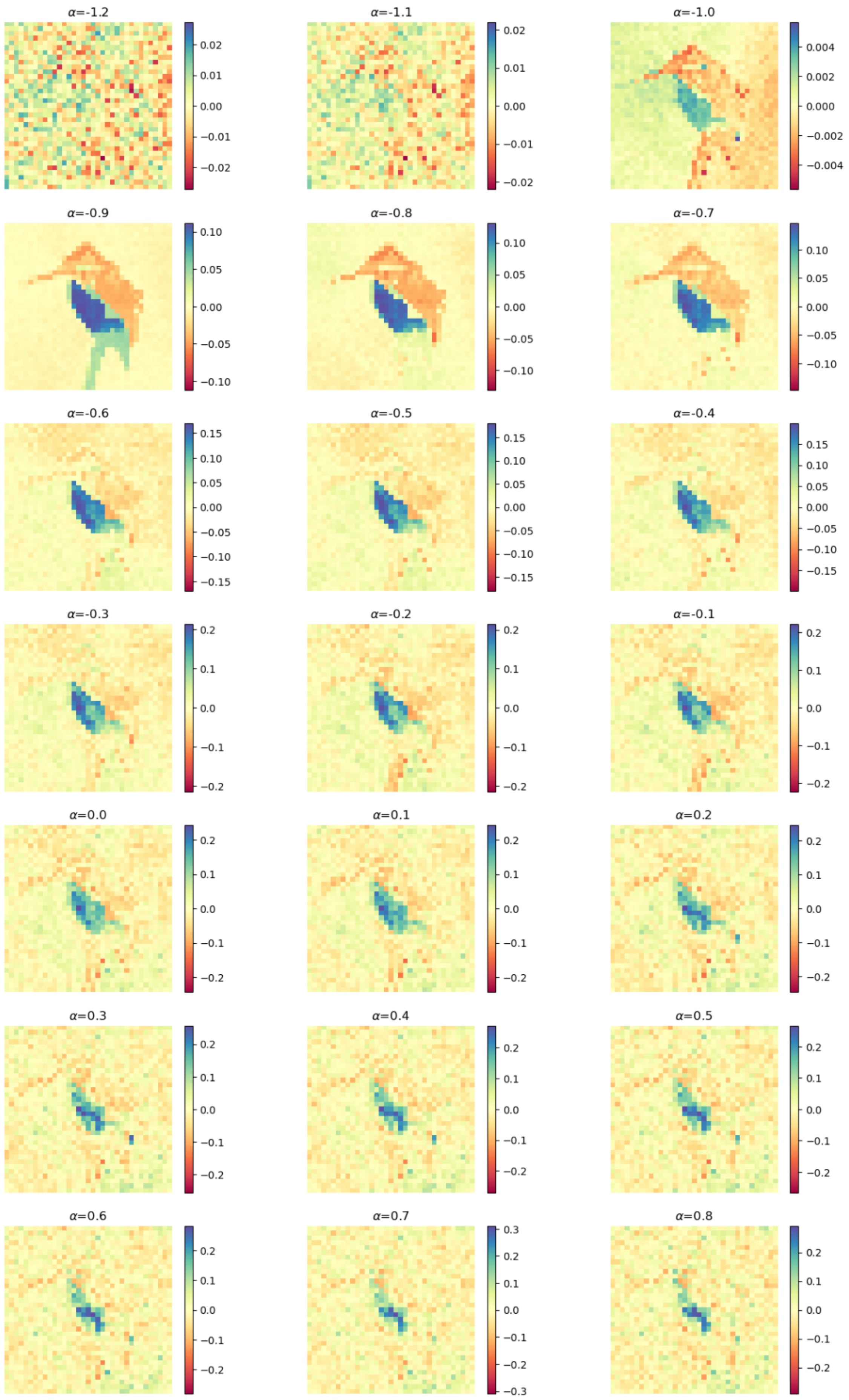
Interpreted as a form of unsupervised learning, the feedback loop pushes the network towards an equilibrium state where the network activity matches the statistical properties of the input patterns.

RESULTS

As shown in the plots below, as the energy stabilises and the network falls into a steady state, certain alpha conditions allows community structures to form which demonstrate similar properties to representations found in GL eigenvectors using principle component analysis. Using the "kingfisher" stimulus, the results show that around -0.9 provides a correlation between low pattern overlap and a high order of patterns detected.



***1** | Maximum overlapped expressed at different variants of alpha. **2** | Number of active patterns. **3** | Energy consumption of a single simulation.*



Simulated pattern overlap and visualisation of active patterns at different variations of alpha.