Spectral Clustering for Identifying Large-Scale Neuron Connectivity

Jackson Cornell [1]

Abstract

Brain-Computer Interfaces (BCI) are becoming increasingly widespread and complex. Algorithms must keep up with this increasing complexity and be able to process and identify patterns in large sets of data being streamed by thousands of electrodes. In this paper, I model the neuronal data as a large graph, and apply spectral clustering algorithms to identify connectivity between simulated neurons of the brain.

1. Introduction

The brain is an amazingly complex structure, which means mapping it is an incredibly arduous task. One method for doing so is by detecting the flow of neuronal spiking, we can map neuron connectivity associated with a certain stimulus or task. By modeling each neuron as a node in a graph, and each neuron connection as an edge, a network of neurons can be mapped from detected spikes. The graph can be subsequently processed by using a technique know as Spectral Clustering to cluster closely-linked neurons, and thus create a map of which groups of neurons fire based on which stimulus or task. It is believed that the underlying mechanics of cognition lies in the synchronized assembly of neuron populations. It can then be reasoned that being able to cluster neuron populations will allow further advancement within the field of neuroscience, as well as the use-cases of brain-computer interfaces (BCIs) for patients with neurological disorders. (Li, 2012) (Paiva et al., 2007)

In this paper, I outline the two subproblems for neuron clustering: finding the functional connectivity between neurons, and applying spectral clustering to find synchronized populations. Two algorithms are given to solve these problems: scale-space projection, and normalized spectral clustering. These algorithms are then implemented and tested on simulated neurons to see how well the algorithms perform in cases where both the number of clusters is known and those where the number of clusters is not known.

2. Problem Statement

The problems that this paper seeks to solve are two fold: one is how to represent neuronal spike trains as functional connections, and the other is how to identify clusters from said connections. In order for accurate clustering of neuron populations, both problems must be adequately addressed.

2.1. Neuronal Functional Connectivity

Spectral clustering necessitates the need for an affinity matrix, or a pairwise distance measure between neurons. The affinity matrix can be thought of as an undirected graph network, where the matrix entries correspond to the vertex weights between nodes. This matrix must be built for neuronal spike trains. Spike trains are a time-dependent measure of the firing of neurons, where the firing, or spike, represents the transmission of information between neurons. Spike trains for a single neuron are represented as a large vector indiced by a time bin from $[0, T]$, where a 1 in index t represents the firing of a spike in that time interval, and 0 indicating the absence of a spike firing. Computing this pairwise distance between neurons is no easy task. The most common approach is to use statistical measures such as correlation, coherence, and synchrony to measure similarities in firing rates (Li, 2012). This method, however, elicits an implicit coarse time scale, controlled by the binning of neural activity (Paiva et al., 2007).

To ascribe a mathematical goal to the problem, some non-linear mapping from the spike train of P neurons recorded over an interval T must be found such that $\{0,1\}^{P\times T} \mapsto \mathbb{R}^{P\times P}$, where 1 represents the firing of a spike from neuron p at time index t and 0 otherwise. The result $W_{ij} \in \mathbb{R}^{P\times P}$ represents the pairwise similarity between two neurons i and j. As we will see in Section 2.2, this is a desirable way to represent our data.

One of the challenges of computing a similarity matrix for spike trains is choosing a time scale. Too small, and it misses the ”big picture”, too large and it won’t be fine enough to differentiate between neuron clusters. One way to solve this issue is to use several time scales to find similarities between neurons. One transform method that works ideal for this sort of problem is the discrete wavelet transform (DWT). The DWT is a way in which a signal can be recursively decomposed into multi-resolution frames of itself. They are similar to the famous discrete Fourier transform (DFT), but offer resolution in both the time and frequency domains. By computing the statistics of multiple timescales, a more accurate picture of the signals' statistics can be found. (Olkkonen, 2011)

2.2. Clustering on Graphs

Clustering has a wide array of uses, ranging from statistics, computer science, biology to social sciences or psychology. Clustering is an unsupervised learning method that, given data, finds groups of similarity between the data. Generic clustering methods generally finds a pairwise distance $d(x_i, x_j)$ between data points i and j. Common distance measures includes the Euclidean distance $||x_i - x_j||_2$, the Manhattan distance $||x_i - x_j||_1$, or the cosine measure $\frac{x_i^T x_j}{||x_i|| \cdot ||x_j||}$. (Rokach & Maimon, 2005)

Standard clustering techniques do have their shortcomings, however. For example, datasets where the groups can not be linearly separated can not be accurately found using k-means or kNN. Another structure that pops up often in the sciences and engineering is that of the graph network. a graph $\mathcal{G}(V, E)$ is represented by its vertices (or data points) V and its edges (or distances between points) E. The adjacency matrix W_{ij} represents the weighted connection between nodes i and j, and is used as an efficient (though sometimes sparse) mathematical representation of the graph $\mathcal{G}$. Another useful mathematical construct for a graph is its degree matrix D_i, where each entry along its diagonal represents the degree, or number of edges, connected to a given edge i. Together, these two matrices can be used to construct the spectral representation of the graph $\mathcal{G}$, called the graph Laplacian matrix L. (Luxburg, 2007)

$$L = D - W \tag{1}$$

$$L = D^{-1/2} W D^{-1/2} \tag{2}$$

Given in Equations 1 and 2, respectively, are the unnormalized and normalized constructions of L. As the name implies, L roughly corresponds to the Laplacian s domain common with differential equations. The matrix L has several important properties that allow it to act as a spectral representation of $\mathcal{G}$, namely:

- For every vector $f \in \mathbb{R}^n$, the following is true: $f'Lf = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2$
- L is symmetric and positive semi-definite
- The smallest eigenvalue of L is 0 and the corresponding eigenvector is the vector of ones $\mathbf{1}$
- All eigenvalues are non-negative and real-valued such that $0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$

(Luxburg, 2007)

There are two main ways to cluster on a graph: spectral methods and minimum cut methods. Minimum cut methods take the following approach: find a partition of the graph $\mathcal{G}$ such that the weighted edges between clusters are minimized, and the weighted edges within a cluster are maximized (Luxburg, 2007). The simplest formulation for partitioning on an adjacency matrix W into k clusters of sub-graphs $\mathcal{A}_k$ is given in Equation 3.

$$\text{cut}(\mathcal{A}_1, ..., \mathcal{A}_k) = \frac{1}{2}\sum_{i=1}^{k} \sum_{j \in \mathcal{A}_i, l \notin \mathcal{A}_i} w_{jl} \tag{3}$$

Minimum cut problems unfortunately tend to be NP-hard, and so the solutions are usually approximated. In fact, spectral clustering methods can be combined with minimum cut methods to produce a relaxed but more efficient solution (Luxburg, 2007). For example, the relaxed version of the the Ratio Cut algorithm is given in Equation 4.

$$\min_{H} \text{Tr}(H^T L H) \text{ subject to } H^T H = I \tag{4}$$

Here, $H \in \mathbb{R}^{n \times k}$ is the indicator matrix. Even though these are relaxed versions of their original formulation, they are still NP-hard. Spectral clustering algorithms, on the other hand, can be computed efficiently and are still highly interpretable. Most come down to the same formulation, with a few numerical tricks for better performance: compute the first k eigenvectors of the graph Laplacian L and use k-means to cluster the eigenvectors. (Luxburg, 2007)

Interestingly, there is a strong link between spectral clustering and kernel methods such as kernel principle component analysis (kernel PCA). Kernel PCA uses the kernel trick to produce a non-linear map of the input data into a reproducing kernel Hilbert space (RKHS), represented by the Gram kernel matrix K. Here, the principle eigenvectors are used to find a lower dimensional, non-linear embedding of the input data. Likewise, spectral clustering works by first finding an embedding of the data, akin to the mapping to RKHS. The principal eigenvectors are then used as the data to perform clustering on. With spectral clustering, the graph Laplacian matrix L can be though of as an analog to the kernel matrix K. In fact, researchers have noted that Algorithm 2 is nearly equivalent to kernel PCA with a clustering step, with the only difference being the normalization of the eigenvectors U. (Ng et al., 2002) (Bengio et al., 2004)

3. Algorithms

3.1. Scale-Space Projection

A population of P neurons will be represented as a spike train, which we will denote $s_p(t)$, the challenge then becomes transforming this high-dimensional and time-varying data as a graph that spectral clustering can be performed on. To do this, we will first perform a wavelet decomposition of $s_p(t)$, which effectively results in each neuron's spike train being decomposed into J separate timescales s_p^j. each entry of s_p^j can be thought of as the probability of neuron p firing at time j, effectively giving a probabilistic representation of the data.

Next, it is important to decompose this large-dimensional data into a $P \times P$ similarity matrix W that represents the interactions of neurons. First, a $P \times P$ correlation matrix Σ_j is computed for each timescale. Σ_j is then compressed into a $PP \times 1$ vector, to form a $PP \times J$ matrix R. Singular Value Decomposition (SVD) is then applied to R, and the first Q dominant modes of the SVD are kept. W is then constructed by summing the Q $P \times P$ matrices, weighted by their corresponding eigenvalue, thus resulting in a $P \times P$ matrix that can be interpreted as a graph of the neuronal connections. (Eldawlatly et al., 2009) (Eldawlatly et al., 2010)

Algorithm 1 Scale-Space Projection

```
  **Input:** Spike train $s_p(t)$, number of wavelet timescales $J$, number of SVD modes $Q$
  **Output:** adjacency matrix $W$
  $s_p^j(t) = W_j s_p(t)$ for $j = 1, .., J$ and $p = 1, ..., P$, where $W_j$ is the Haar wavelet matrix for timescale $j$
  $\Sigma_j = \text{cov}(s_p^j(t))$ for $j = 1, .., J$
  $r_j = \text{vec}(\Sigma_j)$ where $r_j \in \mathbb{R}^{PP \times 1}$
  $R = [r_1 r_2 ... r_J]$ where $R \in \mathbb{R}^{PP \times J}$
  Perform SVD decomposition $U \Sigma V^T = R$
  Let $U_Q \Sigma_Q V_Q^T$ be the first Q modes of the SVD
  Reconstruct $\hat{R} = U_Q \Sigma_Q V_Q^T$ where $\hat{R} \in \mathbb{R}^{PP \times Q}$
  Decompose $\hat{R}$ into $Q$ matrices such that $\hat{R}_q \in \mathbb{R}^{P \times P}$ for $q = 1, .., Q$
  Return adjacency matrix $W = \sum_{q=1}^{Q} \hat{R}_q$
```

(Eldawlatly et al., 2009)

3.2. Spectral Clustering

The theory behind spectral clustering is that the eigenvalues (otherwise known as the spectrum) of a given graph reveals underlying structures. We are given the weighted adjacency matrix W, where each row/column represents a vertex, and each entry in the matrix represents the weighted edge between the vertices. Additionally, the degree matrix D is a diagonal matrix where each entry along the diagonal is simply the number of edges connected to the corresponding vertex. Using W and D, we can construct the Laplacian matrix $L = D - W$, whose eigenvalues holds the spectral information of the graph. Alternatively, a normalized version of the Laplacian can be computed as $L = D^{-1/2} W D^{-1/2}$. (Luxburg, 2007) (Butler & Chung, 2006)

There are two main categories of spectral clustering algorithms used in the literature: those who consist of traditional k-means or kNN clustering schemes, and those who consist of minimum cut clustering schemes. We will focus on the former group, of which nearly all algorithms consist of the same basic procedure. A matrix U is constructed from the eigenvectors of the Laplacian matrix L. The rows t_i of the matrix U are then used as the data points for which a clustering algorithm is applied to. The procedure for the algorithm I will be implementing, Normalized Spectral Clustering, is given in Equation 2.

Algorithm 2 Normalized Spectral Clustering

```
  **Input:** Weighted matrix $W$ of size $P \times P$ and number of clusters $K$
  **Output:** Cluster membership for each neuron
  Construct edge matrix $D$ from $W$
  Construct symmetric Laplacian matrix L
  Compute $U = [u_1 ... u_K] \in \mathbb{R}^{P \times K}$, where $U$ is the first $K$ eigenvectors of L
  Construct $T \in \mathbb{R}^{P \times K}$ where each member in $T$ is defined as $t_{ij} = \frac{u_{ij}}{(\sum_k u_{ik}^2)^{1/2}}$
  Let $t_i \in \mathbb{R}^K$ be a vector corresponding to the $i^{th}$ row of $T$ for $i = 1, ..., P$
  Apply K-means clustering where $t_i$ is the input data
```

(Ng et al., 2002)

4. Experiments

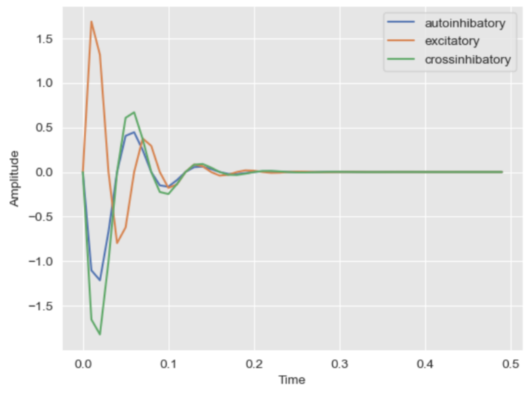
The experiment was run in an incremental fashion; algorithms were first tested on smaller data sets with known solutions to test for validity, before being applied to large data sets. Hyperparameter tuning was used to find optimal algorithm parameters.

4.1. Neuronal Simulation

The basic approach taken for neuronal simulations was to use a stochastic model of neuron firings. The probability that a given neuron has fired increases after the occurrence of a connected neuron fires. This is realized by a decaying sinusoidal wave, where the amplitude is based off the fact that the neuron connection is inhibitory or exhitatory (Eldawlatly et al., 2009). That is, does the occurrence of a spike firing decrease or increase the chance of the following neuron firing. An example of these weight functions are in

Figure 1, where time 0 is the instance a neuron fires, and the probability that the following neuron fires is a function of the amplitude. A weight function $\alpha(t)$ is given in Equation 5.

Figure 1. Weight function for neuron connectivity



$$\alpha(t) = A\sin(\frac{F\pi t}{M})\exp(\frac{-Bt}{M}) \quad (5)$$

The probability of a neuron firing $\lambda_p(t|H_p(t))$ is based off a Poisson process, where $H_p(t)$ denotes the past history of the process (Eldawlatly et al., 2009). We will also define the function $I(t)$ as an indicator function equal to 1 when the previous neuron has fired, and 0 otherwise. The time-dependent probability, otherwise known as the conditional intensity function, is given in Equation 6.

$$\lambda_p(t_i|H_p(t_i)) = \exp(\beta + \sum_{m=1}^{M} \alpha(m\Delta)I(t_i - m\Delta)) \quad (6)$$

where β is the log of the background firing rate (that is, the probability of firing when no stimulus is present), and Δ is the time bin, or sampling period (Eldawlatly et al., 2009). The assumption here is that there is a chain of neurons, where the stimulus of one neuron is based off the firing of only the previous neuron. This is a simple, yet accurate modelling of neuron firing. By creating different chains of neurons, each with different initial stimuli, separate neuronal populations can be simulated. An example initial stimulus is given in Figure 2, along with the resulting firing of the neuron based off the above equations in Figure 3.

Figure 4 shows a raster plot, or a series of spike trains from 10 separate neurons. The first 4 neurons are in one cluster, and the other 6 neurons are in another. The plot should make it clear the dynamics of the neurons. Note the delay between subsequent neuron firings, and how the firings begin to get closer and closer together (that is, the neuron firing rate increases).

Figure 2. Stimulus

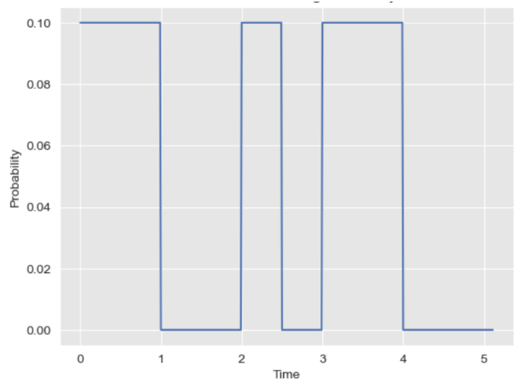
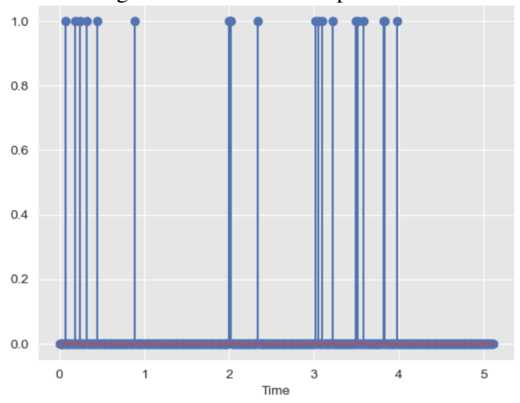


Figure 3. Initial neuron spike train



4.2. Functional Connectivity between Neurons

The next step was to verify the performance of Algorithm 1 in computing the functional connectivity between neurons. We will further verify its use in Section 4.3, but for now we can directly look at the values of the adjacency matrix to see if connected neurons have higher-valued weighted edges than those that aren't connected. Figure 5 shows the computed adjacency matrix from Algorithm 1. Note that neurons 0-3 have higher values between one another than with neurons 4-9, and likewise for neurons 4-9.

4.3. Clustering on Large Neuron Populations

The basic idea behind the simulation of large neuron populations was to recreate the real-world scenario of clustering neuronal recordings in which the user has no prior knowledge. To do this, several clusters were randomly generated, wich the true parameters saved for use in validation. Hyperparameter tuning was carried out to find the parameters that would maximize the silhouette score, with an additional rand index and mutual information score calculated to test against the true clusterings.

The simulation was carried out by drawing the neuronal

Figure 4. Raster plot of two clusters of neurons

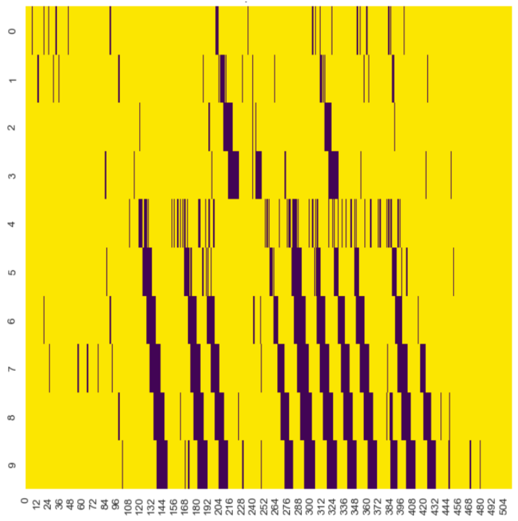
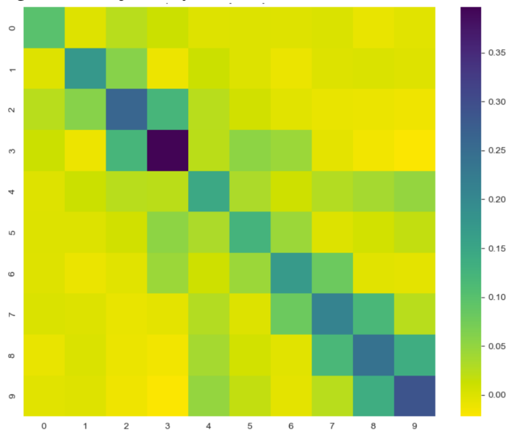


Figure 5. Adjacency matrix of two clusters of neurons



parameters from different uniform densities. Although the parameters wouldn't be too far apart from different neuron populations, they are separate enough for the clustering algorithm to identify separate clusters. First, the number of clusters K was drawn, and then for each cluster, the rest of the parameters were drawn separately. Below lists the parameters and the $pdf's$ they are drawn from, with square brackets referring to a discrete uniform distribution, and parenthesis referring to a continuous uniform distribution:

- Number of clusters $K \sim \mathbb{U}[4, 10]$
- Cluster neuron count $P \sim \mathbb{U}[3, 20]$
- Neuron timescale $M \sim \mathbb{U}(80, 600)$
- Source neuron firing probability $\beta \sim \mathbb{U}(0.1, 0.7)$
- Stimulus 1 start $n_1 \sim \mathbb{U}[0, 300]$
- Stimulus 1 end $n_2 \sim \mathbb{U}[300, 1000]$
- Stimulus 2 start $n_3 \sim \mathbb{U}[1200, 1600]$
- Stimulus 2 end $n_4 \sim \mathbb{U}[1600, 2000]$

Figure 6 shows an example simulation's raster plot. 7 clusters are generated, each with varying amounts of neurons and parameter values.. Figure 7 shows its resulting adjacency matrix. It is worth noting that the clustering algorithm performed much better on more densely packed spike trains, as sparse spike trains were often grouped together.

Figure 6. Raster plot of large-scale neuron simulation

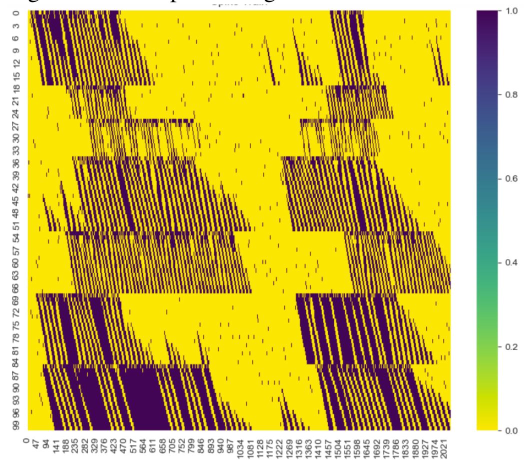
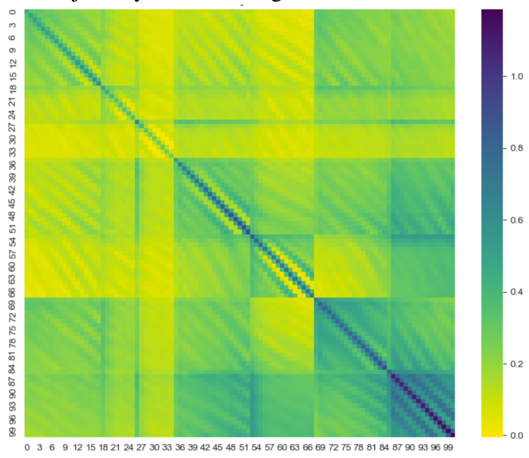


Figure 7. Adjacency matrix of large-scale neuron simulation



Hyperparameter tuning was then carried out to find the optimal parameters. The process was carried out in a manner in which the user has little prior knowledge of the data at hand, with the only assumptions being the range of possible cluster counts and neuron populations being driven by some unknown stimulus to produce sufficiently dense spike trains. The list of parameters tested are as follows: number of clusters $K = [3, ..., 9]$, number of timescales $J = [2, ..., 9]$, the number of dominant modes Q were always set to J, and if the computed L was normalized or not.

The supervised metrics of success used were the rand index, which measures the similarity between the computed and true clusters, and adjusted mutual information, which is similar to the rand index but takes into account differences in cluster count. Additionally, two unsupervised metrics were used: the silhouette score, which measures both the compactness of the clusters as well as their separation, and the elbow method, a heuristic for determining the optimal K. The silhouette score can be calculated along with the rand index and mutual information after each iteration, with values close to -1 indicating poor clustering and values close to 1 indicating good clustering. The elbow method, on the other hand, takes into account the entire range of K's tested, and calculates the distortion and inertia at each iteration. These measures at the plotted "elbow" (the point where distortion and inertia begin to change linearly) estimates the best K. The elbow method proved largely unreliable with the given data.

Empirically, several observations were made:

- The model with the highest rand index was much closer to the true number of clusters K than that with the highest mutual information
- Simulations with sparser spike trains tended to perform worse than those with more dense spike trains
- The estimated number of clusters K tended to be less than the true number, but usually only by 1 or 2
- With denser spike trains, both rand index and mutual information scores tended to be between 0.7 and 0.9
- With sparser spike trains, the scores tended to range from 0.5 to 0.7, with the true cluster count K rarely being found
- The normalized L always outperformed the non-normalized L
- $J, Q \geq 5$ performed the best clustering wise, which is expected as they capture more temporal dynamics

One of the problems this paper aimed to tackle is choosing an optimal cluster count K without performing hyperparameter tuning. One method that showed promise was the eigengap metric (Luxburg, 2007). The basic idea behind the eigengap metric is the the first K eigenvalues are small, but the $K + 1$ eigenvalue is large, thus a "gap" is present, indicating the number of clusters. In practice, however, this proved useless, due to significant overlap in the different cluster embeddings. Alternatively, one can look at the adjacency matrix, such as the one in Figure 7, to see how many "blocks" there are along a given axis. This method only works when there is an implicit ordering in the neurons, which is unlikely to be the case in real-world scenarios. Nonetheless, hyperparameter tuning proved very reliable given sufficiently dense spike trains, with the number of clusters K generally being off by only 1 or 2.

To quantify these empirical observations, a Monte Carlo simulation was carried out. 500 different neuron simulations were performed, with hyperparameter tuning carried out on each. Table 1 shows the calculated statistics.

Measure	Percentage
Silhouette Score Mean	0.158
Silhouette Score Std. Dev.	0.256
$k_{true} - k_{silhouette}$ Mean	-0.058
$k_{true} - k_{silhouette}$ Std. Dev.	2.400
Rand Index Mean	0.715
Rand Index Std. Dev.	0.158
$k_{true} - k_{rand}$ Mean	-0.18
$k_{true} - k_{rand}$ Std. Dev.	1.391
Mutual Information Mean	0.786
Mutual Information Std. Dev.	0.124
$k_{true} - k_{mutual}$ Mean	-0.138
$k_{true} - k_{mutual}$ Std. Dev.	1.511

Table 1. Statistical results from Monte Carlo simulation

All algorithms and simulations were built from scratch with python in the Jupyter Notebook environment. The code used to run these experiments can be found on GitHub. [1]

5. Conclusion

In this paper, I presented a systematic way to identify coupling between individual neurons given their spike trains, and then used spectral clustering to identify clustered populations of said neurons. The Scale-Space Projection algorithm is reviewed and implemented as a way to model the neurons as a graph $\mathcal{G}$ using multiple time scales and subsequent eigenvector projection. The Normalized Spectral Clustering algorithm is then presented as a way to identify clusters of neurons by processing directly on $\mathcal{G}$ using its adjacency matrix W. Additionally, the connection is made between spectral clustering and kernel PCA, as the presented Normalized Spectral Clustering algorithm is simply k-means performed on the kernel PCA transform of the input data.

The identifying of optimal hyperparameters K, J, and Q for the aforementioned algorithms are then discussed, using hyperparameter tuning to search the possible space of parameters. Large scale simulations of neurons showed that,

[1] https://github.com/jcornell616/spectral-clustering-for-identifying-large-scale-neuron-c

with little to no prior knowledge and sufficiently dense spike trains, the silhouette score is an adequate metric for determining these optimal parameters. Several other metrics, the elbow method and eigengap method, are tried and deemed through empirical trials to be ineffective.

Future research on this topic can focus on finding a more effective method for determining the number of clusters K, as hyperparameter tuning is computationally intense. Additionally, finding an alternative to the Scale-Space Projection algorithm that is robust to sparse spike trains can increase the use-case for these methods. The paper has shown there are effective ways to blindly find clusters of neuron populations, which may pave the way for exciting BCI applications such as being to better understand the underlying machinations of cognition in the human mind.

6. Acknowledgments

Thank you Dr. Huang for the fun semester. I really enjoyed the topics and felt like I learnt a lot. I hope to be able to apply may of the things I was taught in this class in industry after I graduate.

References

Bengio, Y., Delalleau, O., Roux, N. L., Paiement, J.-F., Vincent, P., and Ouimet, M. Learning eigenfunctions links spectral embedding and kernel pca. *Neural computation*, 16(10):2197–2219, 2004.

Butler, S. and Chung, F. *Handbook of Linear Algebra*. CCR Press, 2nd edition, 2006.

Eldawlatly, S., Jin, R., and Oweiss, K. G. Identifying functional connectivity in large-scale neural ensemble recordings: A multiscale data mining approach. In *Neural Computation 21*, pp. 450–477, 2009.

Eldawlatly, S., Zhou, Y., and Oweiss, K. G. On the use of dynamic bayesian networks in reconstructing functional neuronal networks from spike train ensembles. In *Neural Computation 22*, pp. 158–189, 2010.

Li, L. *Kernel based machine learning framework for neural decoding*. University of Florida, 2012.

Luxburg, U. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.

Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing 14*, pp. 849–856, 2002.

Olkkonen, H. *Discrete Wavelet Transforms: Biomedical Applications*. BoD–Books on Demand, 2011.

Paiva, A. R., Rao, S., Park, I., and Príncipe, J. C. Spectral clustering of synchronous spike trains. In *2007 International Joint Conference on Neural Networks*, pp. 1831–1835. IEEE, 2007.

Rokach, L. and Maimon, O. Clustering methods. In *Data mining and knowledge discovery handbook*, pp. 321–352. Springer, 2005.