Predicting Neural Progenitor Cell Networks

Nicolas Grandel

***Abstract*—Predicting neuronal networks from calcium fluorescence data is extremely difficult due to the nature of the data collection. Current methods to overcome this difficulty, while powerful, are somewhat arbitrary, presenting new difficulties in later analysis. Specifically, current ad-hoc methods often make arbitrary distinctions between connected and unconnected neurons. In this project, I attempted to use supervised and unsupervised learning techniques to work around this problem, with varying degrees of success. Of particular interest was the success of the K-Means algorithm, which replicated the success of current methods while avoiding some of their pitfalls.**

I. Introduction

One of the most interesting (and least understood) aspects of developmental biology is the growth of networks in neurons. The ways in which pre-neurons (known as neural progenitor cells, or NPCs) form and cultivate inter-cellular communication networks is an active topic in research that is extremely difficult to quantify. This is in large part due to the nature of how cell communication data is gathered. The most common form this information takes is fluorescence microscopy images of cells stained with fluorescent proteins that react to the presence of Ca^{2+}, an ion that is used in both chemical and electrical neuro-transmission.[1] While valuable, this data is incredibly noisy, and as a result it is often very difficult to truly quantify whether or not a given signal is related to any other given signal.[2]

In order to account for this, rather than comparing image intensity values directly, researchers often treat a time series of intensities as a vector of "spikes" and "not spikes" (known as spike trains), and then measure the lagged cross-correlation between vectors, often using an arbitrary cut-off value for differentiating spikes and for determining correlated signals. This ad-hoc procedure casts an enormous shadow over research in this area.[3] For my project, I have attempted to compare several different classification algorithms, including K-Nearest Neighbors, Logistic Regression, and Support Vector Machines, against the traditional ad-hoc method, where I give these algorithms summary features of two of the calcium spike trains and they give a binary response to whether those two trains are connected. To do so, I have attempted to generate artificially noisy signal data, with examples known to be correlated or not correlated, and allowed the algorithms and ad-hoc method to classify these signal sets, comparing the results at the end. I have also attempted to approach the problem from a different angle, by starting from the assumption that every neuron was a member of one hidden network, and using a clustering algorithm (specifically K-Means) to sort the neurons into their respective networks by giving the algorithm the set of raw spike trains and report the network each train belongs to.

While this may be an area of niche research, nearly every paper using the ad-hoc method discusses the need for a more reliable alternative. Having already used this method myself, I can personally confirm this need, and hope to further aspects of research I have already participated in through this project.

II. Related Work

Throughout the past several years, there have been a variety of approaches to alleviating this problem of inferring network connections. Many of them have focused on replacing the "full correlation" method described above with a more robust method. In particular, two methods have come to the fore: 1) using Bayesian inference[3],[4] to predict network connections and 2) using a method known as "partial correlation" to remove the influence of other signals on the measured correlation.[5],[6] The use of Bayesian networks is promising, and gets around the problem of differentiating between unconnected and connected neurons. The use of the "partial correlation" method, however, runs into the same problem as using the "full" cross-correlation method described above. It is only recently that this exact issue been addressed by creating a "scrambled" signal by taking a given signal f, selecting a random time point t, and creating a new signal $f_{scrambled}$

such that $f_{scrambled}[1:N] = [f[t:N]f[1:t-1]]$ (Figure 1).[7]

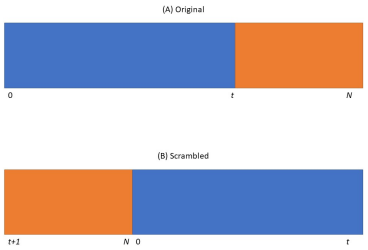

Fig. 1. Example of scrambled signal (B) generated from original signal (A).

The cut-off is then selected as the mean of the set of correlations for all pairs of scrambled sets. While this does provide a reasoned method for removing insignificant correlation values, it also tends to be dependent on the true network density, resulting in over- or under-estimating the cut-off value for networks that are more or less connected, respectively.[2] As a result, a method for creating a decision boundary is still required when using a correlation-based method.

III. Dataset and Features

The data for this project was gathered in 2016 at Rice University in the Qutub Lab, thanks to the work of Arun Mahadevan. This specific dataset was comprised of 1000 time-lapsed calcium fluorescence microscopy images over the course of 15 minutes (Figure 2). Images were converted to gray scale, and then the average pixel intensity values at each time point for 53 cells were gathered from these images, which were then transformed into spike trains, where the value of the train at time point t was 1 if the difference between the intensity value at time t and time $t-1$ was greater than one standard deviation of the entire signal. Summary statistics, including total number of spikes, spike frequency, and the period of spikes, were also calculated to be used in supervised learning methods.

Because "true" networks were difficult to discern, a sample set of signals that were known to be connected or unconnected were first generated. This signal set was created by calculating the mean frequency of spikes in the actual signal set, and then generating several random connected networks of signals by setting each network's frequency as a Gaussian distribution, and then randomly shifting signal sets forward and backwards.

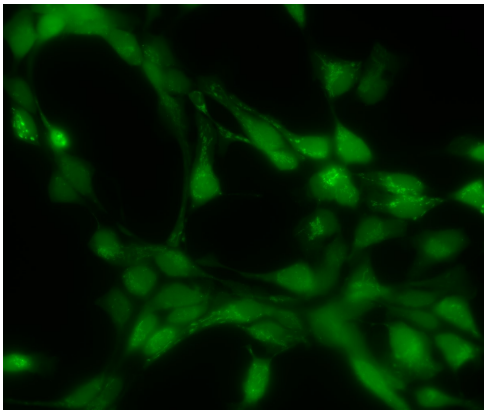

Fig. 2. Example frame of dataset.

IV. Methods

A. Supervised Methods

In my efforts to create a new method for determining a cut-off correlation value, I attempted to use three different supervised learning methods: K-Nearest Neighbors, Logistic Regression, and Support Vector Machines, each implemented in the sklearn library.[8]

The first method, K-Nearest Neighbors, was simply the standard algorithm, in which a given member of a test set is assigned the same class as the k-closest members of the training set. In this case, the simple Euclidean distance algorithm was used as a measure of distance.

The second method, Logistic Regression, takes the training data, where the input is represented by the vector X and the output by the vector Y, and attempts to maximize the log likelihood function:

$$l(\theta) = \sum_{i=1}^{M} y^{(i)} \log h(x^{(i)}) + (1-y^{(i)}) \log(1-h(x^{(i)})),$$

where $h(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$. To do so, the algorithm uses the stochastic gradient ascent rule:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}.$$

Finally, the last algorithm, the Support Vector Machine, proves to be the most complicated of the three. SVMs perhaps make the most intuitive sense for this problem, since it is this method that actually creates a decision boundary by maximizing the distance between the decision boundary and the

members of either class. It does this by maximizing the equation

$$W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle,$$

such that $0 \leq \alpha_i \leq C, i = 1, ..., m$ and $\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$.

B. Unsupervised Methods

After the efforts above met a dead end, I then attempted to approach the problem from a different angle, by starting from the assumption that every neuron was a member of one hidden network, and using a clustering algorithm (specifically K-Means) to sort the neurons into their respective networks by giving the algorithm the set of raw spike trains and report the network each train belongs to.

This implementation of K-Means was self-implemented, to ensure that the proper metric was used to measure the distance between examples. However, besides using the lagged cross-correlation as the distance metric, this implementation of K-Means was not overly unique. Much like most implementations, this version takes the given data and randomly picks k examples to be cluster centers. It then iteratively assigns each data point to the closest cluster center, then makes the new center of each cluster the "average" of all of the members of that cluster. This implementation only changes the meaning of "average" by making the new cluster center the example with the highest mean cross-correlation with all other members of the cluster.

V. Results

A. Supervised Learning

For my first attempt at solving this problem, I implemented the supervised learning algorithms described above using the sklearn libraries. I then created the synthetic signal training set with a similar number of cells to the test set (N = 50) and exactly the same number of images (M = 1000). I also split this synthetic data up into training and validation sets, and then used the real signal data as a test set, using the connectivity created by the standard cross-correlation method as the class for each neuron-pair. In this first iteration, I only gave the algorithms the spike train itself and the mean time between spikes, thinking that this would be enough information to extract the connectivity of the neuron network. However, as can be seen from the results above, there was significant room for improvement.

Trial 1.

Model	Train Accuracy	Validation Accuracy	Test Accuracy
Log	0.891	0.885	0.645
SVM	0.874	0.872	0.633
KNN	0.881	0.878	0.642

After several rounds of trying different feature inputs, I eventually settled upon the addition of spike frequency and the standard deviation of time between spikes, which resulted in the results seen in the table labeled Trial 2. It was at this point that I realized that the artificial spike set I had generated was not capturing the essence of the real-world data, and that accurately predicting networks in the artificial space would not equate to doing the same in the real-world space.

Trial 2.

Model	Train Accuracy	Validation Accuracy	Test Accuracy
Log	0.953	0.908	0.734
SVM	0.923	0.884	0.752
KNN	0.896	0.851	0.728

B. Unsupervised Learning

Once I came to that conclusion, I decided that I would approach the problem from a different angle. Instead of testing for the connectivity of every pair of neurons, I would instead find the underlying networks by clustering the signals based on their cross-correlation. With this in mind, I implemented the K-Means algorithm using cross-correlation as a distance metric, as described above. After one simple implementation, I found that this method accurately recreated the networks found using the traditional cross-correlation method, and even excluded certain cells which were found to be unlikely to be members of the network. Below, the inter-cluster correlations and a graphical representation of the clusters can be seen (Figure 3,4).

Trial 3. Inter-cluster cross-correlation.

Model	Train	Test
K-Means	0.885	0.842
Cross-Corr	0.862	0.837

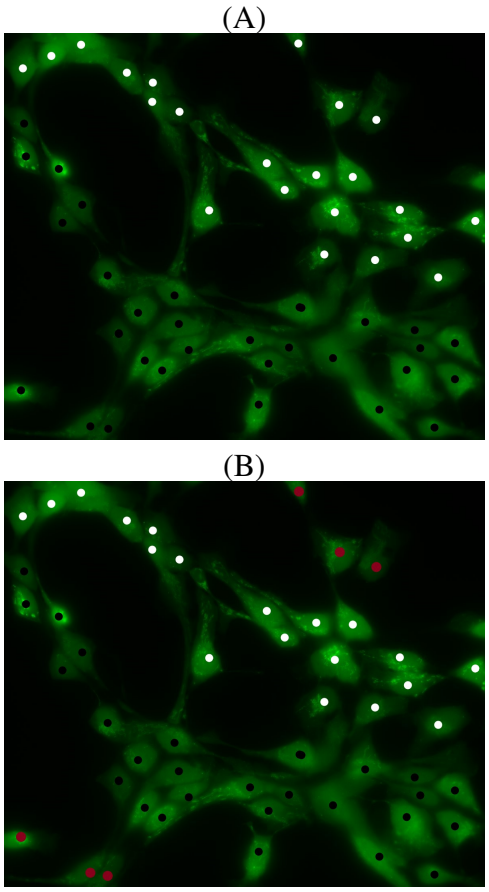

Fig. 3. Networks generated by (A) cross correlation method and (B) K-Means. Similar color indicates belonging to same network. Notice how certain edge neurons were eliminated in the K-Means result.

VI. Conclusion

While none of the supervised learning algorithms ended up performing well enough to replace the cross-correlation method, it is nonetheless interesting to consider why certain algorithms (the SVM in particular) performed better than others. It seems likely to me that the SVM was able to learn/capture certain features of the data that the Logistic Regression and K-Nearest Neighbors algorithms were not. Of greater interest is the performance of the K-Means algorithm. It not only seemed to recreate the networks found by the traditional ad-hoc cross-correlation method, but also improved upon it slightly by excluding certain edge neurons from the networks. Considering the lower average correlation of the neuron signals found within this data set, this implies that the K-Means algorithm manages to overcome the dependency of the ad-hoc method on the nature of the dataset in question. If I had more time and resources, I would certainly continue to explore this, as well as other unsupervised learning algorithms, for the potential they have to give us powerful insights into these neuronal networks.

References

[1] Gobel, W., & Helmchen, F. (2007). In Vivo Calcium Imaging of Neural Network Function. *Physiology*, 22(6), 358. https://doi.org/10.1152/physiol.00032.2007

[2] Rahmati V, Kirmse K, Markovi D, Holthoff K, Kiebel SJ (2016) Inferring Neuronal Dynamics from Calcium Imaging Data Using Biophysical Models and Bayesian Inference. *PLoS Comput Biol.* 12(2): e1004736. https://doi.org/10.1371/journal.pcbi.1004736

[3] Sporns, O. (2014). Contributions and challenges for network models in cognitive neuroscience. *Nature Neuroscience*, 17, 652.

[4] Mumford, J. A., & Ramsey, J. D. (2014). Bayesian networks for fMRI: A primer. *NeuroImage*, 86(Supplement C), 573582. https://doi.org/https://doi.org/10.1016/j.neuroimage.2013.10.020

[5] Smith, S. M., Miller, K. L., Salimi-Khorshidi, G., Webster, M., Beckmann, C. F., Nichols, T. E., Woolrich, M. W. (2011). Network modelling methods for FMRI. *NeuroImage*, 54(2), 875891. https://doi.org/https://doi.org/10.1016/j.neuroimage.2010.08.063

[6] Marrelec, G., Krainik, A., Duffau, H., Plgrini-Issac, M., Lehricy, S., Doyon, J., & Benali, H. (2006). Partial correlation for functional brain interactivity investigation in functional MRI. NeuroImage, 32(1), 228237. https://doi.org/https://doi.org/10.1016/j.neuroimage.2005.12.057

[7] Smedler, E., Malmersj, S., & Uhln, P. (2014). Network analysis of time-lapse microscopy recordings. *Frontiers in Neural Circuits*, 8, 111. http://doi.org/10.3389/fncir.2014.00111

[8] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 28252830.