Spike Train Analysis

Trinity College Dublin

Cathal Cooney

5 12 2014

Abstract

Dedication

Declaration

Acknowledgements

Contents

Chapter 1

Introduction

more to follow

Chapter 2

Clustering methods in Spike Train Analysis

2.1 Networks

I wanted to approach neuroscience from the point of view that since there are clearly connections between neurons in the same region of the brain, that these connections could possibly illuminate the functional landscape of the brain. This point of view prompted the study of some Network Theory, a branch of Mathematics closely related to Graph Theory. Network Theory differs from Graph Theory in that it deals with more life-like networks and the traits that one would expect to see in such networks. The main difference between these would be the connectivity profile of the network; in Graph Theory connections are usually spread somewhat uniformly through the graph, whereas in Network Theory we would expect communities to form.

A mathematical network is simply a collection of nodes and links between these nodes. An undirected network with n nodes can be completely described by the adjacency matrix A_{ij} where $A_{ij} = 1$ if nodes i and j are connected, and is equal to zero otherwise. In this case $A_{ij} = A_{ji}$ since two nodes are either connected or not. If we allow the A_{ij} to take values other than one and zero then we can call it a weighted network, and if we allow the matrix to not be symmetric, then we get a directed network, where $A_{ij} = 1$ if there is an arrow pointing from node j to node i. We will deal almost exclusively with binary undirected networks, as the theory is richer in this case, and often one can say that two objects, or properties, are related or that they are not.

The part of Network Theory that we feel could be useful for Neuroscience is clustering measures, and algorithms to maximise these measures. The main focus is on the measure known as the *modularity*.

2.1.1 Modularity

The *modularity* is a measure of how much more prominent intra-cluster links are than inter-cluster links in a given clustering.

Thus, the modularity is a measure of a clustering of a network that assesses how much more community structure there is in the network than there would be in a random network with similar properties, such as degree of nodes. If the probability of a link between nodes i and j is P_{ij} and g_i is the community to which node i belongs, then it can be described as [**?**]:

$$Q = \frac{1}{2m} \sum_{i,j} [A_{ij} - P_{ij}] \delta(g_i, g_j) \tag{2.1}$$

where δ is the Kronecker delta, so $\delta(g_i, g_j) = 1$ if nodes i and j are in the same cluster, and zero otherwise. m is the total number of links in the network, and so $\frac{1}{2m}$ is just a normalising factor. The modularity of a cluster is then the number of edges within a cluster minus the expected number of edges.

We need to understand what exactly P_{ij} should be. In undirected networks, we should get $P_{ij} = P_{ji}$. Since the case of all nodes being in the same cluster is not interesting, and displays no further community structure, we set $Q = 0$ in that case. Therefore, we get $\sum_{i,j}[A_{ij} - P_{ij}] = 0$, and,

$$\sum_{i,j} P_{ij} = \sum_{i,j} A_{ij} = 2m, \tag{2.2}$$

where m, as before, is the total number of edges in the network, so if the degree of node i is k_i, then

$$m = \frac{1}{2} \sum_i k_i = \frac{1}{2} \sum_{i,j} A_{ij}. \tag{2.3}$$

Beyond this, there is more scope for choice. We could suppose that each node in the "random" network has degree equal to the average degree of the network, but this ignores local structure in the network. If we suppose that our "random" network should keep the same degree for all nodes instead then we get the definition of Newman as in [**?**]. So, if the expected degree of each vertex is equal to the actual degree of the vertex, then:

$$\sum_j P_{ij} = k_i. \tag{2.4}$$

If we suppose now that beyond this constraint that edges are placed at random, then the probability of two nodes connecting is dependent on just their degrees. Supposing the probabilities for each end of a single edge is independen t, a reasonable assumption in a large network, where all the degrees are small relative to the total number of edges, we get that $P_{ij} = f(k_i)f(k_j)$ for some function f on their degrees. Then, by equation 2.4, we get

$$\sum_{j=1}^{n} P_{ij} = f(k_i) \sum_{j=1}^{n} f(k_j) = k_i, \tag{2.5}$$

so $f(k_i) = Ck_i$, for some constant C, and we get

$$2m = \sum_{i,j} P_{ij} = C^2 \sum_{i,j} k_i k_j = (2mC)^2, \tag{2.6}$$

so, $C = 1/\sqrt{2m}$ which gives us our probability P_{ij} as

$$P_{ij} = \frac{k_i k_j}{2m}, \tag{2.7}$$

and the modularity is

$$Q = \frac{1}{2m} \sum_{i,j} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(g_i, g_j). \tag{2.8}$$

We get values between -1 and 1 for any clustering. Since we know that not dividing the network into separate clusters gives a modularity of zero, this gives us a lower bound for “good” clusterings.

The benefit of finding a good clustering can be seen in Figure **??**; a good clustering can reveal what the nature of the network is, rather than it being a mess of nodes and links. While the modularity itself is a measure of a given clustering of a network, the maximum modularity is a property of the network itself, which can tell us about the inherent community structure of the network. To actually determine what clustering will give the maximum modularity we would require an exhaustive search which becomes unfeasible for networks of a moderate to large size, and so we need to use clustering algorithms to maximise the modularity.

2.1.2 Newman's Eigenvalue Algorithm

In [?] Newman noted that if we supposed that the network were split into two clusters, then we could redefine the modularity in terms of a quadratic form. We define a vector s, which keeps track of the split,

$$s_i = \begin{cases} 1 & \text{if node } i \text{ in cluster 1} \\ -1 & \text{if node } i \text{ in cluster 2} \end{cases}$$

Then we can redefine modularity as:

$$\begin{aligned} Q &= \frac{1}{2m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right)\frac{1}{2}(s_i s_j + 1) \\ &= \frac{1}{4m}\sum_{i,j}\left(A_{ij} - \frac{k_i k_j}{2m}\right) s_i s_j, \end{aligned} \tag{2.9}$$

Now, if we define a matrix $\mathbf{B}$ where $B_{ij} = A_{ij} - k_i k_j / 2m$, which we call the *modularity matrix*, then we can rewrite this equation as:

$$Q = \frac{1}{4m}\mathbf{s^t B s} \tag{2.10}$$

Now, since $\mathbf{B}$ is an $n \times n$ symmetric matrix, we know that it has n real eigenvalues λ_i, with corresponding eigenvectors u_i. Now, we order the λ_i so that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$, and we write

$$\mathbf{s} = a_1\mathbf{u_1} + \ldots + a_n\mathbf{u_n}.$$

The equation for the modularity becomes

$$\begin{aligned} Q &= \frac{1}{4m}\sum_i a_i \mathbf{u}_i^t \mathbf{B} \sum_j a_j \mathbf{u}_j \\ &= \frac{1}{4m}\sum_i \lambda_i (\mathbf{u}_i^{\mathbf{t}}.\mathbf{s})^2 \end{aligned} \tag{2.11}$$

so, the positivity of the modularity depends completely on the λ_i, in particular the most positive eigenvalue λ_1. This means that to maximise the modularity we should take our "split vector" $\mathbf{s}$ to be as close to the first eigenvector $\mathbf{u}_1$ as possible. To do this, we choose $\mathbf{s}$ as follows:

$$s_i = \begin{cases} 1 & \text{if } u_{1_i} > 0 \\ -1 & \text{if } u_{1_i} \leq 0 \end{cases} \qquad (2.12)$$

This gives us a clustering of the network into two smaller clusters. Of course, this split is not strictly in the direction of λ_1, but it is our best possible estimate[1], given two clusters. If the modularity of the split is not positive, then we reject the split and say that the best clustering is to leave the network as it is. Similarly, if the first eigenvalue $\lambda_1 = 0$ then we say that the best clustering is to leave the network undivided, as we know that the modularity matrix $\mathbf{B}$ always has a zero eigenvalue, with eigenvector $\mathbf{u} = (1, 1, \dots, 1)$, which corresponds to no split of the network.

With the network split into two smaller clusters, the next question is how to split the network further. We look at the clusters one-by-one and split them until it gives no benefit to the modularity of the clustering. One could naively think that we should look at the adjacency matrix of the cluster itself, and form the corresponding modularity matrix, but this ignores the connectivity of the overall network. We could inadvertantly split the subgraph into communities which would make sense within the subgraph, but would ignore the connections from outside the subgraph. Instead, we need to use the original modularity matrix $\mathbf{B}$ and calculate what difference a split of the subgraph would make to the modularity.

We call the subgraph g and the *modularity contribution* ΔQ of a split of g is:

$$\Delta Q = \frac{1}{2m}\left[\frac{1}{2}\sum_{i,j\in g} B_{ij}(s_i s_j + 1) - \sum_{i,j\in g} B_{ij}\right]. \qquad (2.13)$$

This is the term in the modularity of the network that a split $\mathbf{s}$ of g would change, minus the modularity of leaving the subgraph g whole. We can get our "subgraph modularity matrix" $\mathbf{B}^{(g)}$ now:

$$\Delta Q = \frac{1}{4m}\left[\sum_{i,j\in g} B_{ij} s_i s_j - \sum_{i,j\in g} B_{ij}\right]$$

[1]In fact, it has been noted from well-known networks, such as Zachary's karate club [**?**] that the absolute value of the ith entry of $\mathbf{u}_1$ gives a good indication of the "strength" of the membership of node i to its cluster [**?**].

$$\begin{aligned}
&= \frac{1}{4m}\sum_{i,j\in g}\left[B_{ij} - \delta_{ij}\sum_{k\in g} B_{ik}\right] s_i s_j \\
&= \frac{1}{4m}\mathbf{s^t}\mathbf{B}^{(g)}\mathbf{s} \qquad (2.14)
\end{aligned}$$

and so $\mathbf{B}^{(g)}$ that takes the value

$$B_{ij}^{(g)} = B_{ij} - \delta_{ij}\sum_{k\in g} B_{ik} \qquad (2.15)$$

for labels i, j of nodes in the subgraph g.

We can use the same approach as before and find the most positive eigenvalue of $\mathbf{B}^{(g)}$ to determine our favoured split $\mathbf{s}$ to maximise ΔQ. $\mathbf{B}^{(g)}$ still has the property that each of its rows and sum to zero, so we still have a zero-eigenvalue that represents no split of the subgraph. This can now be the stopping criterion for the algorithm; if we get a zero-eigenvalue we say that the subgraph is indivisible and stop splitting it up. However, we must be careful that as the most positive eigenvalue gets smaller, relative to its negative eigenvalues, there may be no benefit to the modularity of accepting such a split. In this case, we check, by calculating ΔQ for the proposed split, to see if it is worth continuing splitting the subgraph. If we get $\Delta Q \leq 0$ then we stop, otherwise we continue splitting the graph. The algorithm is summarised in algorithm 1.

This spectral method for maximising the modularity has shown remarkably good results [**?**], in fact considerably better results than the *betweenness* algorithm of Newman and Girvan [**?**], despite the apparent drawback of always splitting the network/subgraph in two parts. It is also possible to use the first m eigenvalues and use the ith entry of the eigenvectors as coordinates in $\mathbf{R}^m$ [**?**], but this then requires a choice of number of clusters, which may not be known from the data. This may also require calculating many more eigenvalues of the large matrix $\mathbf{B}$, which can be computationally expensive. I plan to try and implement a similar algorithm using the first two eigenvectors, λ_1 and λ_2, as coordinates in $\mathbf{R}^2$ and using k-means clustering on the projections of these coordinates on S^1. This method will of course lead to greater computation times, but it should hopefully resolve some of the problems of splitting the network in two, without becoming completely prohibitive computationally.

Algorithm 1 This is Newman's eigenvalue algorithm for maximising the modularity of a network.

```
  Calculate the modularity matrix $\mathbf{B}$, where $B_{ij} = A_{ij} - k_i k_j / 2m$
  Find the most positive eigenvalue $\lambda_1$ and its eigenvector $\mathbf{u}_1$
  if $\lambda_1 = 0$ then
    Stop the algorithm with no split in the network.
  else
    Split network such that node $i$ in first group if $u_{1_i} > 0$ and in second
    group otherwise.
Ensure:    Modularity of split $> 0$
  end if
  for all Subnetworks $g$ do
    Calculate $\mathbf{B}^{(g)}$, where $B_{ij}^{(g)} = B_{ij} - \delta_{ij} \sum_{k \in g} B_{ik}$
    Find the most positive eigenvalue $\lambda_1^{(g)}$ of $\mathbf{B}^{(g)}$ and its eigenvector $\mathbf{u}_1^{(g)}$.
    if $\lambda_1^{(g)} = 0$ then
      Subnetwork can split no further.
    else
      Split network such that node $i$ in first group if $u_{1_i}^{(g)} > 0$ and in second
      group otherwise
Ensure:      $\Delta Q > 0$ for split of subnetwork $g$.
    end if
  end for
```

2.2 Spike Trains

Neurons convey information through the nervous system by generating electric pulses which propagate along nerve fibers. These pulses are called action potentials or spikes [**?**].

A neuron has a resting potential of approximately -70mV relative to its surroundings, and we say that the cell is polarised at this point. The membrane potential of a neuron becomes less negative with current flowing into it, but will tend back towards its resting potential unless the membrane potential nears a certain threshold. If a neuron is depolarised so that its membrane potential is raised above this threshold, the neuron generates an action potential. An action potential, or spike, is a 100mV fluctuation in the membrane potential which lasts approximately 1ms. Following the production of a spike the neuron briefly becomes hyperpolarised, and as such cannot generate a spike for the next couple of milliseconds. This period, when a spike cannot be fired, is called the absolute refractory period of a neuron. Since the neuron becomes hyperpolarised by the spike, there is also a period in which it is "more difficult" for the neuron to spike again, this is called the relative refractory period of a neuron.

Spikes are very important because they propagate over long distances without attenuation, and so are the way in which neurons communicate with one another throughout the nervous system. The timing of spikes is of particular importance; for example, in motor neurons when a number of spikes happen close together it leads to movement of the muscle, the more spikes that happen together, the faster the muscle twitches. While there is some variation in the duration, amplitude and shape of action potentials, we can largely treat them as identical processes, where the timing is the only consideration. So, if a neuron spikes n times in a certain trial, then the trial can be described by the times of the spikes t_i. Then we can describe the *spike train* mathematically as:

$$s(t) = \sum_{i=1}^{n} \delta(t - t_i). \tag{2.16}$$

The δ here is the Dirac delta, which essentially gives a spike of volume one at the point t_i. It can be very difficult to differentiate between different spike trains, and so we have *metrics* to tell them apart.

2.2.1 Spike Train Metrics

If we look at spike trains from a specific neuron, we would like to be able to "decode" them and be able to describe the stimulus which evoked them. Unfortunately, this seems like a very difficult problem, so we first settle for trying to distinguish which spike trains amongst a collection were evoked by the same stimulus. To do this, there are several *metrics* in the literature which tell us the "distance" between two spike trains.

In Mathematics a metric space is a space where we can always tell the "distance" between two elements, whether or not we can give the "coordinates" of the elements in the space. A metric is a function on a set X, $d : X \times X \to [0, \infty)$ that basically follows our intuition for what a distance should be. That is, distances are non-negative, symmetric and only zero if two elements are the same. The Triangle Inequality states that you should never be able to shorten the distance between two points by going through an intermediate point, so it is the notion that the shortest distance between two points is a straight line.

2.2.2 Van Rossum metric

The metric that we primarily use is the metric described by van Rossum in [?], which is simply based on the L^2 metric on function spaces. If we view the spike trains as a collection of Dirac delta functions, then we can get a function by convolving the distribution $s(t)$ as above with a kernel. A common kernel that is used is the exponential kernel

$$k(t) = \begin{cases} \frac{1}{\tau} e^{-\frac{t}{\tau}}, & t \geq 0 \\ 0, & t < 0 \end{cases} . \tag{2.17}$$

This kernel has some advantages, like causality and computability, over a gaussian kernel. So, we get a new function $u(t)$ after convolving the spike train with the kernel:

$$u(t) = s * k(t) = \int_0^T s(t-s) k(s) \, ds \tag{2.18}$$

Figure **??** above shows the form of the function that we get when we convolve a spike train with the exponential kernel. Once we have these functions for different spike trains, u and v say, then we calculate the distance between them by taking the L^2 metric on the function space, that is:

$$d(u,v) = \sqrt{\int_0^T (u(t) - v(t))^2 \, dt} \tag{2.19}$$

This metric has some mathematical advantages over other metrics, such as Victor- Purpura [?], an "edit-length" metric, because there is a lot of material in functional analysis on the L^2 metric. It performs to a similar standard to most other metrics, and remarkably the choice of kernel has little effect on its efficiency.

2.2.3 *k*-Medoids Clustering

By defining a metric on spike trains, we can tell how close different responses are to each other, but there are no "co-ordinates" in the metric space which would help with clustering responses. The standard method of clustering spike-trains is similar to k-means clustering, but since there is no standard method for computing a "mean spike-train" given a number of spike trains, it must be altered slightly.

For k-means clustering, we select k random points in the space in which clustering is occuring, then each point in the space would be in the cluster of the closest of these k points to itself. Then we would find the mean of all the points in a cluster to get a new "centre" for the cluster and re-cluster each point to the closest of these k means. This process is repeated until the means are stable, which gives a clustering of the points into k clusters.

Unfortunately, since there is no mean in the metric space of spike-trains, we cannot perform k-means clustering, rather, we perform k-medoids. In k-medoids, we randomly choose k points in the data, which we call medoids, and then we place each other point of the data in the cluster of the "closest" (or, least dissimilar) medoid to it. Then, for each cluster, we swap the medoid for each other point in the cluster, and choose the one with the least overall distance as the new medoid. This step is shown in Figure **??** below. We repeat these steps until there is no change in the medoids. This method suits us for spike-trains, as the idea of a "mean" spike-train is not well understood, but there is currently a very promising method proposed by other members of this lab, which is to be submitted shortly.

There are some downsides to using k-medoids, that I hoped could possibly be improved by using network methods. The primary disadvantage is that you must select how many clusters in advance, and hence you must know how many different stimuli were used when you look at a data set. The next chapter deals with our efforts to address this issue.

2.2.4 Clustering Responses with Modularity

We proposed that we could perhaps improve on current methods of sorting responses by using the modularity clustering algorithm. The advantage to such a method would be that we would hopefully not need to know how many different stimuli there were. Since the algorithm determines when to stop itself, it could simply be run to completion. Such a method could perhaps be used for many different data sets, when the number of stimuli was not known, to determine the different responses.

We used the data set from [?], which featured spike trains from anesthetized adult male zebra finches as they listened to natural birdsong. Each neuron was played 20 songs ten times each, so we had 200 data points from which to form a network for each neuron.

We formed a network by first taking the Van-Rossum metric distances between each pair of responses, then we chose a threshold value τ for the network and we set

$$A^{\tau}_{ij} = \begin{cases} 1 & \text{if } d(i,j) < \tau \\ 0 & \text{otherwise} \end{cases} . \tag{2.20}$$

Once we have the adjacency matrix for our network, we can run the algorithm to maximise modularity for the network. For the data that we used, each spike train was less than a second long, so the maximum Van Rossum distance between any two trains was one, so we ran the algorithm for different threshold values τ between zero and one incrementing τ by steps of size 0.001.

As we can see in Figure **??**, the profile of the graph of the maximum modularity versus the threshold looked promising, as in nearly all cases it had a clear maximum, but unfortunately the clusters of responses bore little resemblance to the stimuli, and usually had too few clusters.

It appears that the margins are just too small, and there is no natural network, rather a bunch of networks determined by a strict cut-off, so the modularity algorithm cuts some of the correct clusters in two. A recent paper by Humphries [?] uses all of the positive eigenvalues to cluster the responses with more accuracy. This method is rather computationally expensive, and uses even the very small positive eigenvaluse, which tell us very little about the positivity of modularity, so we tried to use Newman's more simple algorithm, despite its obvious drawbacks.

The thing that was most disappointing was that the correct clustering nearly always gave a lower modularity than the algorithm, so it seems like network clustering methods just may not be very useful for this problem.

2.3 Discussion

Due to the fact that the modularity that the algorithm found for the network is higher than the modularity of the correct clustering, it is clear that network methods are not useful to sort stimuli from a single neuron. Since the original goal was to illuminate the functionality of the brain, or at least small areas of the brain, this non-result doesn't matter too much. We want to get simultaneous recordings from many neurons, and form networks which would correspond to different neurons rather than spike trains.

There are different ways to form the networks of many neurons, but we will mainly be interested in neurons that seem to *drive* other neurons. That is, neurons whose firing seems to improve the probability of the other neuron firing significantly. A method that particularly interests us is the method of *Incremental Mutual Information* of Singh and Lesica [**?**], which tries to reduce the uncertainty of a neuron as much as possible before checking whether another neuron influences it.

Any network formed from this measure would have to be directed, so we need a method to turn a directed network into an undirected network. Newman describes a very neat way to do this in [**?**], where two nodes are related in the undirected network by how many commons nodes they point to in the directed network. He calls this the *bibliographic coupling* of two nodes. In a brain network, this could give us a "map" of information flow.

Once we have our new networks, we may use some other network measures to examine them as in [**?**]. The *clustering coefficient* is a useful local measure for how clustered a node is, and it may be useful to use this along with the modularity to find and study community structure.

Chapter 3

Spike Train Metrics

The SPIKE distance proposed by Kreuz et al in [?] is an instantaneous parameter-free distance measure between spike-trains. By instantaneous measure, it is meant that for each time t there is a time-local distance between two spike-trains. This measure can the be integrated over the length of the spike-trains to give a parameter-free distance measure between spike trains. In the study of spike-train metrics, often a multi-unit measure is desired (to minimise compounding any errors that may result from inefficient spike-sorting?). A multi-unit measure is a distance between collections of labelled spike-trains. The SPIKE distance in [?] does not lend itself to a simple multi-unit extension, so instead a simpler version of the SPIKE distance is extended.

3.1 Single-unit recordings

With a view to extending the SPIKE distance to a multi-unit distance measure, it is useful to review the definition of the SPIKE measure provided in [?]: Given two spike-trains x and y, where $x = \{t_1^x, \ldots, t_n^x\}$ and $y = \{t_1^y, \ldots, t_m^y\}$, where $t_1^x, \ldots, t_n^x, t_1^y, \ldots, t_m^y$ are the spike times. The SPIKE distance in [?] has the nice property that the distance is bounded such that it is always between zero and one. Unfortunately, to achieve a "natural" extension to a multi-unit measure, this property is sacrificed. A simpler version of the distance, without the strict upper bound of one, is used.

The simpler distance is very quick and easy to calculate. A 'gap' is associated with each spike; this is the distance to the nearest spike in the other spike-train. The total distance between two spike trains is then the sum of these gaps. The original SPIKE distance included additional normalisation factors that have been omitted in simplifying the measure. To form the time-local distance, a weighted sum of the gaps for the corner spikes is made, that is, the spikes preceding and following the time of interest in each of the two spike trains. The weighting is chosen so that the integral of the time-local function is just the sum of the gaps.

First, the gaps are calculated for each spike in the two spike-trains. This is simply the nearest spike in the other spike train:

$$\Delta t_i^x = min_i(|t_i^x - t_i^y|) \tag{3.1}$$

At each time instant, these is a unique set of four corner spikes: the preceding and following spikes from each spike train, which are labelled $t_P^x(t), t_F^x(t), t_P^y(t)$ and $t_F^y(t)$; these are, respectively, the preceding and following spikes in spike-train x and the preceding and following spikes in spike-train y.

For each spike-train, w, a time-local distance is then calculated using the associated gap of the four corner spikes for each spike-train, $w = x, y$:

$$s_w(t) = \frac{\lambda_P(t)\Delta t_P^w(t) + \lambda_F(t)\Delta t_F^w(t)}{I^w(t)} \tag{3.2}$$

where

$$\lambda_F^w(t) = \frac{t - t_P^w(t)}{I^w(t)},\ \lambda_P^w(t) = \frac{t_F^w(t) - t}{I^w(t)} \tag{3.3}$$

and $I^w(t)$ is the size of the interval in which t is contained:

$$I^w(t) = t_F^w(t) - t_P^w(t). \tag{3.4}$$

Now, the time-local distance for each neuron is added to give the overall time-local distance:

$$s(t) = s_x(t) + s_y(t). \tag{3.5}$$

This simplified time-local SPIKE distance has the advantage that its integral is simply the sum of the gaps of each spike; that is:

$$\int_0^T s(t)\,dt = \sum_w \sum_i \Delta t_i^w. \tag{3.6}$$

In practice this simplified version of SPIKE produces similar time profiles to the version described in [?]. The time-local distance profile for two similar spike-trains is given in Figure [NEED TO ADD PICTURE].

3.2 Extension to multi-unit recordings

In the multi-unit case a distance is defined between two multi-unit recordings. Thus, rather than two spike-trains there are two sets of spike-trains; $\mathbf{X} = \{\mathbf{x}_i\}$ and $\mathbf{Y} = \{\mathbf{y}_i\}$, where $i \in 1 \ldots N$ and the index i labels the neuron.

Here the single-unit distance above is extended to a multi-unit distance by including the possibility that the nearest spike for one neuron may belong to a different neuron in the other set, however there is a distance penalty associated with changing from one neuron to the other. This penalty is similar to the "costs" in edit-length metrics such as Victor-Purpura [**?**]. In other words, it is necessary to introduce a parameter k, which quantifies a fictional distance between spikes fired in different cells. The size of this distance quantifies the importance of the labelling of the neurons, and varying k interpolates smoothly from $k = 0$, a summed population (SP) code, to k large, a labeled line (LL) code. While theoretically the LL code occurs as $k \to \infty$, in practice a LL code occurs when k is on the order of $2/\lambda$, where λ is the average frequency of the trials. The gaps can then be calculated as follows:

$$\Delta t_\alpha^{\mathbf{x}_i} = \min_{\beta,j} \left(|t_\alpha^{\mathbf{x}_i} - t_\beta^{\mathbf{y}_j}| + k\left[1 - \delta(i,j)\right] \right) \tag{3.7}$$

where $\delta(i,j)$ is the Kroneker delta. Hence, if the spikes have the same label in $\mathbf{x}$ and $\mathbf{y}$ there is no added distance, otherwise k is added to the time difference between the spikes. This is illustrated in Figure 3.1:

The time-local distance measure is then defined as before, using the corner spikes normalised by the inter-spike-interval:

$$s_{\mathbf{X}}(t) = \sum_{\mathbf{x}_i \in \mathbf{X}} \frac{\lambda_{\mathrm{P}}(t)\Delta t_{\mathrm{P}}^{\mathbf{x}_i}(t) + \lambda_{\mathrm{F}}(t)\Delta t_{\mathrm{F}}^{\mathbf{x}_i}(t)}{I^{\mathbf{x}_i}(t)} \tag{3.8}$$

where

$$\lambda_{\mathrm{F}}^{\mathbf{x}_i}(t) = \frac{t - t_{\mathrm{P}}^{\mathbf{x}_i}(t)}{I^{\mathbf{x}_i}(t)} \tag{3.9}$$

and

$$\lambda_{\mathrm{P}}^{\mathbf{x}_i}(t) = \frac{t_{\mathrm{F}}^{\mathbf{x}_i}(t) - t}{I^{\mathbf{x}_i}(t)} \tag{3.10}$$

and $I^{\mathbf{x}_i}(t)$ is the length of the interval in the spike-train $\mathbf{x}_i$ containing t, $I^{\mathbf{x}_i}(t) = t_{\mathrm{F}}^{\mathbf{x}_i}(t) - t_{\mathrm{P}}^{\mathbf{x}_i}(t)$. As before:

$$s(t) = s_{\mathbf{X}}(t) + s_{\mathbf{Y}}(t) \tag{3.11}$$

Once more, this distance measure has the nice property that if the time-local measure is integrated over the course of the trial it equals the sum of the gaps of each spike:

$$\int_0^T s(t)\,dt = \sum_{\mathbf{W}} \sum_{\mathbf{w}_i \in \mathbf{W}} \sum_{\alpha} \Delta t_\alpha^{\mathbf{w}_i} \tag{3.12}$$

3.2.1 Testing on data

The multi-unit distance measure was tested on similar test data to that found in Houghton & Sen [**?**], where two Poisson neurons form a receptive field and are each connected to two leaky integrate-and-fire neurons with relative strength a and $1 - a$. Hence, for $a = 0$ each receptive neuron is connected to a single LIF neuron, and for $a = 0.5$, each LIF neuron receives input equally from each of the receptive neurons.

3.3 ISI distance

3.3.1 Single unit recordings

With a view to extending the ISI distance to a multi-unit distance measure, it is useful to review the previous definition of the ISI distance between two spike trains $\mathbf{x}$ and $\mathbf{y}$. The ISI distance can be thought of as a rate-comparison, where the inter-spike interval (ISI) is used as a proxy for the firing rate. As with all the time-local distance measures, it is defined as the interval over time of a local distance function $s(t)$.

Given two spike-trains $\mathbf{x}$ and $\mathbf{y}$, then $I^{\mathbf{x}}(t)$, $I^{\mathbf{y}}(t)$ are the inter-spike intervals at time t, for the spike-trains $\mathbf{x}$ and $\mathbf{y}$. The time-local distance function is then defined in Kreuz et al. (2007) as:

$$s(t) = \begin{cases} 1 - I^{\mathbf{x}}(t)/I^{\mathbf{y}}(t) & \text{if } I^{\mathbf{x}}(t) \leq I^{\mathbf{y}}(t) \\ 1 - I^{\mathbf{y}}(t)/I^{\mathbf{x}}(t) & \text{otherwise} \end{cases} \tag{3.13}$$

From the point of view of generalising to more than one neuron, it is convenient to rewrite this as:

$$s(t) = \frac{|I^{\mathbf{x}}(t) - I^{\mathbf{y}}(t)|}{\max(I^{\mathbf{x}}(t), I^{\mathbf{y}}(t))} \tag{3.14}$$

3.3.2 Initial extension to multi-unit recordings

In the multi-unit case a distance is defined to be between two multi-unit recordings. Thus, rather than two spike-trains there are two sets of spike-trains; $\mathbf{X} = \{\mathbf{x}_i\}$ and $\mathbf{Y} = \{\mathbf{y}_i\}$, where $i \in 1 \dots N$ and the index i is labelling the neuron.

The ISI-distance is a rate-based metric, so here a similar approach to multi-unit recordings as was used successfully for the van Rossum metric, another metric calculated using estimated rates, is used. That approach, outlined in Houghton and Sen (2008) [**?**], replaces the rate with a rate-vector in an N-dimensional space. To do this, each neuron is assigned a unit vector describing its direction in the rate-space. At a given time, this is multiplied by the estimated rate of the neuron to give a rate-vector. The population rate-vector is then the sum of individual rate-vectors for each of the neurons in the population.

An L^1 norm, $|I^{\mathbf{x}}(t) - I^{\mathbf{y}}(t)|$, appears in the numerator of $s(t)$ in the single-unit case, so the natural extension of this idea to the multi-unit ISI-distance involves an L^1-structure on the N-dimensional space, which is the ISI-space in this case. In practice this means the individual vectors are unit vectors under the L^1-norm; so for example for $N = 2$, one vector may be $(1, 0)$ and the other one would be $(1 - \alpha, \alpha)$. The corresponding individual neuron ISI-vectors would then be:

$$\mathbf{I}^{\mathbf{x}_1}(t) = \begin{pmatrix} I^{\mathbf{x}_1}(t) \\ 0 \end{pmatrix} \tag{3.15}$$

$$\mathbf{I}^{\mathbf{x}_2}(t) = \begin{pmatrix} (1-\alpha) I^{\mathbf{x}_2}(t) \\ \alpha I^{\mathbf{x}_2}(t) \end{pmatrix}. \tag{3.16}$$

The overall population ISI-vector $\mathbf{I}^{\mathbf{x}}(t) = \mathbf{I}^{\mathbf{x}_1}(t) + \mathbf{I}^{\mathbf{x}_2}(t)$:

$$\mathbf{I}^{\mathbf{X}}(t) = \begin{pmatrix} I^{\mathbf{x}_1}(t) + I^{\mathbf{x}_2}(t) - \alpha I^{\mathbf{x}_2}(t) \\ \alpha I^{\mathbf{x}_2}(t) \end{pmatrix} \tag{3.17}$$

It remains to extend the definition of $s(t)$ to ISI-vectors. It is proposed here that this should be:

$$s(t) = \frac{\|\mathbf{I}^{\mathbf{X}}(t) - \mathbf{I}^{\mathbf{Y}}(t)\|_1}{\sum_i \max(I_i^X, I_i^Y)} \tag{3.18}$$

where I_i^X, I_i^Y are the ith components of $\mathbf{I}^{\mathbf{X}}$ and $\mathbf{I}^{\mathbf{Y}}$ respectively, and:

$$\|\mathbf{I}^{\mathbf{X}}(t) - \mathbf{I}^{\mathbf{Y}}(t)\|_1 = \sum_i |I_i^X - I_i^Y| \tag{3.19}$$

One important property of any multi-unit distance measure is that it interpolates from the summed population (SP) code to the labelled line (LL) code . In the case of the labelled line code, it is necessary to consider what the result should be, for example in the Victor Purpura metric the LL code is the sum of the distances of the individual neurons, whereas in the case of the van Rossum metric, the LL code is the Pythagorean sum of the distances. Here the LL code corresponds to when the vectors are all perpendicular, this is $\alpha = 1$ in the $N = 2$ example above. Up to an overall L^1 rotation, this means that the individual ISI vectors will have a single non-zero component, and $s(t)$ is given by:

$$s(t) = \frac{\sum_i |I^{\mathbf{x}_i}(t) - I^{\mathbf{y}_i}(t)|}{\sum_i \max(I^{\mathbf{x}_i}, I^{\mathbf{y}_i})} \tag{3.20}$$

Given the rational structure of the ISI-distance, this seems to be the natural structure for the LL code. Conversely, when all the vectors are parallel, the result is the SP code:

$$s(t) = \frac{|\sum_i I^{\mathbf{x}_i}(t) - \sum_i I^{\mathbf{y}_i}(t)|}{\max(\sum_i I^{\mathbf{x}_i}(t), \sum_i I^{\mathbf{y}_i}(t))} \tag{3.21}$$

in which, effectively, the ISIs are averaged across the population before being compared in the distance measure.

3.3.3 Numerical tests

This multi-unit extension of the ISI distance is tested on the same data that was used to test the multi-unit van Rossum distance in Houghton and Sen (2006). In this simulation, two integrate and fire neurons receive noisy input from two sources. A parameter a determines how much these two inputs are mixed. If $a = 0$ each receives independent input, if $a = 0.5$, each receives an input averaging the two sources. Both neurons also receive shot noise. The two inputs are inhomogeneous Poisson processes, and there are five separate randomly chosen inhomogeneous functions. Each of these are used to drive the neurons 20 times, and the distance function is used to cluster these 100 neurons by stimulus in the usual way. The accuracy of this clustering is evaluated by calculating the Transmitted Information (h) of the clustering. The results are shown in Figures **??** and **??** . Compared to similar graphs in [**?**], the performance is similar to the multi-unit van Rossum metric; but compared to the van Rossum metric, the optimal value of α in the distance measure is less clearly modulated by a.

While the extension proposed above was derived logically from the single-unit case, it does not agree with the standard definition of a summed-population code, that is, it is possible to find a population of neurons which have spiked at exactly the same time, but where the average inter-spike intervals are not equal at a moment in time.

3.3.4 Alternative extensions to the multi-unit case

Upon comparing the above extension to the previous extension to the multi-unit case by Kreuz et al. [**?**], it became clear that there was a fundamental difference in how each distance measure viewed the concept of a rate-based metric.

The extension proposed in [**?**] interpolates between the average of the individual ISI distances and the ISI distance of the two "population neurons", that is, treating the individual spike times from the entire population as though they were all from the same neuron. This is done with a "population parameter" p, which runs from 0 (SP) to 1 (LL). This is given by:

$$s(t) = (1-p)\left(\frac{|I^{\mathbf{x}}(t) - I^{\mathbf{y}}(t)|}{\max(I^{\mathbf{x}}, I^{\mathbf{y}})}\right) + p\left(\frac{\sum_i |I^{\mathbf{x}_i}(t) - I^{\mathbf{y}_i}(t)|}{\sum_i \max(I^{\mathbf{x}_i}, I^{\mathbf{y}_i})}\right), \tag{3.22}$$

with notation as before, and $I^{\mathbf{x}}(t)$ is the interval in $\mathbf{x}$ at time t, where the population is viewed as a single neuron.

The extension proposed above can be compared to the Kreuz extension by replacing the ISI of the "population neuron" with the average of the ISIs across the population of neurons. This leads to the following equation:

$$s(t) = (1-p)\left(\frac{|\sum_i I^{\mathbf{x}_i}(t) - \sum_i I^{\mathbf{y}_i}(t)|}{\max(\sum_i I^{\mathbf{x}_i}(t), \sum_i I^{\mathbf{y}_i}(t))}\right) + p\left(\frac{\sum_i |I^{\mathbf{x}_i}(t) - I^{\mathbf{y}_i}(t)|}{\sum_i \max(I^{\mathbf{x}_i}, I^{\mathbf{y}_i})}\right), \tag{3.23}$$

.

There is a fundamental difference in how each of these distance measures view what a SP metric should be. The Kreuz example believes that there is information being carried by the frequency of arrival of spikes across the population, regardless of origin, and the second measure using the average ISI believes that each neuron is essentially carrying the same message, with an inherent noise due to the biological constraints of neurons. Each argument has its own merits [**I think we should find a few papers where each side of the argument would be given merit by the results**], and it could be useful to have both of these measures, as in equations 3.22 and 3.23, depending on the focus of the experiment.

[**combine the population distance and the average distance graphs into one graph**]

3.4 Adaptive SPIKE & ISI distances

The downside to the measures introduced above is that there must be parameters to vary between the SP and LL cases, and there is no good way to choose a parameter, which often leads to a grid search to find the optimal parameter. A goal of this work is to find a way for the data itself to choose its own population parameter. The SPIKE and ISI distances for single neurons are both parameter-free, so it would be ideal if an extension could be found that was also parameter-free.

Since the ISI distance is a distance based on the firing rate of the neurons, the assumption was made that if the rates of the individual neurons were reasonably similar throughout the two populations, then the correct code to use would be an SP code, whereas if they were very different, then a LL code should be used.

The ISI extensions above have a population parameter p that varies from 0 to 1, so a measure of similarity (or dissimilarity) that varies from 0 to 1 would be ideal. The normalised cross-entropy of the ISIs of the two populations is such a measure. Labelling the ISIs as above, the cross-entropy at any point in time t is calculated as:

$$h_{\mathbf{X},\mathbf{Y}}(t) = \frac{1}{\log 2n} \sum_{\mathbf{z} \in \mathbf{x},\mathbf{y}} \sum_{i} -\frac{I^{z_i}(t)}{\sum_{\mathbf{z}} \sum_i I^{z_i}(t)} \log \left(\frac{I^{z_i}(t)}{\sum_{\mathbf{z}} \sum_i I^{z_i}(t)} \right) \tag{3.24}$$

This is a dissimilarity measure on the ISIs across the two populations. If the ISIs are very similar, then the cross-entropy is close to 1, if they are very dissimilar then it is close to 0. Thus, the cross-entropy is a good candidate for an instantaneous population parameter p, or more accurately $(1-p)$.

Figures **??** and **??** show that this does not work very well, because one would hope that the adaptive measure would perform on a par with the maximum values of transmitted information from figures **??** and **??**, but it is considerably lower than the max values in both the "average" and "population" ISI measures.

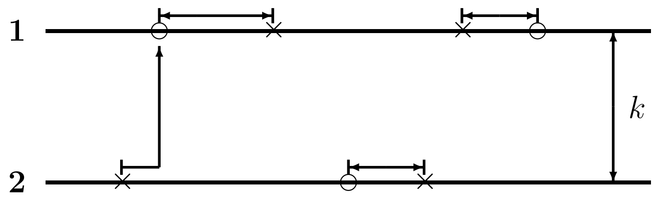

Figure 3.1: An example of how to calculate the gaps between spikes. If the circles represent spikes from **X** and crosses represent spikes from **Y**, then in the picture above, the distance k is simply the cost of relabelling a spike. We also see that the gaps are not necessarily symmetric.

Chapter 4

A simple neuron model

Sparse coding has been observed in neurons in the visual tract [**?**], a simple model is proposed for such neurons.

It is unlikely that a neuron is ever truly “off”, so it is assumed that a neuron has a base-firing state that will be referred to as the “off-state”. For the sake of simplicity, it is assumed that a neuron in such an “off-state” would have a constant firing rate, λ_d. Correspondingly, for sparse-coding a neuron must have a higher firing-rate when the feature that it codes for is present; this firing rate, λ_u, is also taken to be constant.

This idea is simplified to the extreme case, where a neuron is either in its up-state and has a high firing-rate, or it is “off”. The model is treated as a poisson process, for ease of calculation.

The data that is simulated has an “up rate” λ_u and a “down rate” λ_d. For the initial trials, $\lambda_d = 0$. When in the down-state, the state switches “up” with expected frequency u, and when in the up-state it switches “down” with expected frequency d. Thus, the average background rate, r, of the inhomogeneous poisson process can be calculated as:

$$r = \frac{\frac{\lambda_u}{d}}{\frac{1}{u} + \frac{1}{d}} = \frac{u\,\lambda_u}{u + d} \tag{4.1}$$

since the expected time for being in the up-state is simply $1/d$ and the expected time in the down-state is $1/u$.

4.1 Estimating the firing rate $r(t)$

With the model for the firing rate as above, it is possible to explicitly calculate the probability of being in the up-state for the time following a spike. In the initial setting, where λ_d is set to zero, then it is known when there is a spike that the rate is in the up-state, so at the time of spiking t_0 the probability $p(t_0)$ of being in the up-state is equal to one. For the estimate of the rate, $\tilde{r}(t)$, this is reflected by setting $\tilde{r}(t_0) = \lambda_u$. Then, the probability is reset every time there is a spike, so it is only required to calculate the probability of being in the up-state given that there has been no spike since the time t_0 of the last spike.

It is possible, using Baye's Theorem, to calculate a first approximation of the probability of being in the up-state at a time $t + \Delta t$, given a spike at time $t = t_0$ for small Δt.

Let $X =$ up at $t = t_0 + \Delta t$, $Y =$ no spike since $t = t_0$ and $Z =$ spike at $t = t_0$. Then,

$$P(X|Y|Z) = \frac{P(Y|X|Z)P(X|Z)}{P(Y|Z)} = \frac{(1-\lambda_u\Delta t)(1-d\Delta t)}{1-r\Delta t} \tag{4.2}$$

Then, it is possible to calculate the first approximation to the probability of being up at time t, by letting $t_0 = 0$, $\Delta t = t/n$.

$$\begin{aligned} P(\text{up at } t|Y|Z) &= \lim_{n\to\infty}\prod_{k=1}^{n}\frac{(1-\lambda_u t/n)(1-dt/n)}{1-rt/n} \\ &= \lim_{n\to\infty}\prod_{k=1}^{n}\left(1+t\frac{r-\lambda_u-d}{n}\right) \\ &= \lim_{n\to\infty}\left(1+t\frac{r-\lambda_u-d}{n}\right)^n \\ &= e^{(r-\lambda_u-d)t} \end{aligned} \tag{4.3}$$

Recalling from equation 4.1 that $\lambda_u = r(u+d)/u$, get:

$$p = e^{-\frac{d(r+u)}{u}(t-t_0)} \tag{4.4}$$

Chapter 5

Conclusion

more to follow

Appendix A

Appendix

more to follow