Identifying Localized Signals On Graphs

Introduction

In this problem, we are presented with N cells and their expression of each of M genes (or their expression with dropout). So given a gene g, we would like to come up with a way of answering the question: how localized is g on our set of cells? Two problems are: 1. creating spatial embeddings for the cells based off of their gene expression and 2. using these embeddings, quantify the 'spread' of the expression of g over a gene graph $\mathcal{G}$. Two main approaches include: 1. decomposing signals s into a sparse linear combination of diffusion wavelets; by weighing the coefficients of the large scales more heavily, we quantify a sort of "average scale" of the signal. A second approach uses a spatial embedding of the cells and uses the average distance between two points generated from s viewed as a distribution.

1 — Frequency Approach

One possible way to approach is this problem is to decompose the signal s into a linear combination of diffusion wavelets at various scales. Then the more components the signal has in larger scales, the more spread the signal is.

In particular, we define a family of diffusion wavelets like so: suppose we have a diffusion operator M on our graph, with *columns* giving probability distributions from points. For example, if we have spatial data with affinity matrix A and degree matrix D, $M = D^{-1}A$. We define:

$$P = \frac{1}{2}(I + M) \text{ a lazy random walk operator}$$

As is typical, we will define $\Psi_j = P^{2^{j-1}} - P^{2^j}$. Note that for large values of j, $\Psi_j(\delta_i) \approx \Psi_j)(\delta_k)$ if vertex i is near k. Thus, we can reduce the number of columns in Ψ_j by taking some subset of the columns to form a matrix $\tilde{\Psi}_j$ such that the total error in projecting Ψ_j onto $\tilde{\Psi}_j$ is less than some ϵ fraction of the norm of the whole Ψ_j. That is,

$$\frac{1}{\|\Psi_j\|}\|^2 \|\Psi_j - \tilde{\Psi}_j(\tilde{\Psi}_j{}^T\tilde{\Psi}_j)^{-1}\tilde{\Psi}_j^T\Psi_j\|^2 \leq \epsilon \tag{1}$$

Think of $\tilde{\Psi}_j$ as a selection of wavelets at scale j that span the graph. In general, $|\tilde{\Psi}_j|$ decreases in j and increases in ϵ. Here's an algorithm for obtaining such a $\tilde{\Psi}_j$. For example,

Algorithm 1 Obtaining $\tilde{\Psi_j}$

Input: A set of diffusion wavelets Ψ_j and a tolerance $\epsilon \geq 0$
Ouput: $\tilde{\Psi_j}$, whose columns are a subset of Ψ_j such that they nearly span Ψ_j
Let $\tau \leftarrow \epsilon \|\Psi_j\|^2$
$Q, R, p \leftarrow Pivotal - QR(\Psi_j)$
$R_{norm} \leftarrow \|\Psi_j\|^2$
while for i in $1...n$ **do**
  $R_{norm} \leftarrow R_{norm} - \sum_{j=i}^{n} R_{i,j}^2$
  if $R_{norm} \leq \tau$ **then**
    return $\Psi_j(p(1), p(2)...p(i))$
  end if
end while

This algorithm is proven to be correct in the Appendix. If n is very large, we could also input a randomly sampled set of columns into the algorithm. Think of Ψ_j as being some spanning set of wavelets. Of course, for larger and larger scales, we might imagine that we can take relatively few of these, as they can cancel a great deal of low frequency signals. Here is such an example:

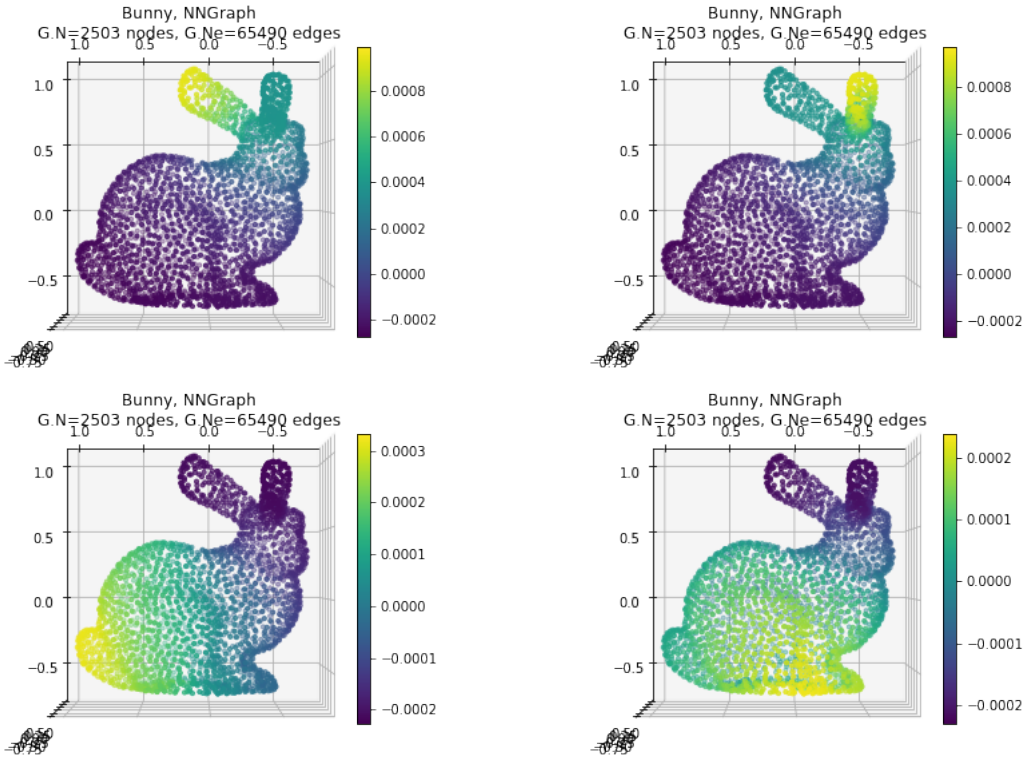

Figure 1: Example of $\tilde{\Psi_j}$. Notice that the original matrix Ψ_j is 2503×2503. In contrast, $\tilde{\Psi_j}$ is 2503×23. Four such "spanning" columns as outputted by Algorithm 1 using tolerance $\epsilon = 10^{-3}$ are shown above.

Detecting Local Signals

With this out of the way, we can present a possible algorithm for detecting "localized" signals. We first create a matrix $\Phi = \begin{bmatrix}\Psi_0 & \Psi_1..\Psi_J\end{bmatrix}$. That is, it is the horizontal concatenation of all of our wavelets. We then define a weight vector $\omega = \begin{bmatrix}1...1 & 2...2 & .... & 2^J...2^J\end{bmatrix}$. For a more efficient implementation, we could also set Φ using the $\tilde{\Phi}_j$'s instead; this would also help avoid overfitting. Regardless, normalize the columns of Φ for consistency. Now, given a signal s (assume it is normalized), we can estimate a sparse linear combination x of columns of Φ that approximate s. For our purposes, let's employ the matching pursuit algorithm. Recall that this algorithm attempts to solve, given some ϵ,

$$\min_x \|x_0\| \text{ such that } \|\Phi x_0 - s\| < \epsilon$$

So suppose we have an algorithm $MP(\Phi, s, \epsilon)$ that outputs the desired x. We can then set a measure of locality ℓ of s like so:

$$\ell(s; x) = \omega \cdot |x| = \sum_i |x_i|\omega_i = \sum_j 2^j \sum_{k:k \text{ corresponds to scale j}} |x_k|$$

Which motivates the following algorithm for finding localized signals:

Algorithm 2 Calculating Smoothness

Input: A diffusion operator M and signal s defined on a set $\mathcal{X}$. SPARSE, a boolean variable encoding whether to use the full Ψ_j or $\tilde{\Psi}_j$, an ϵ_1 to dictate $\tilde{\Psi}_j$. OMP a routine for matching pursuit. And associated tolerance ϵ_2 for this algorithm.
Output: $\ell(s)$, a measure of localization of s on the set $\mathcal{X}$.

$P \leftarrow \frac{1}{2}(I + M)$
$J \leftarrow \log(|\mathcal{X}|)$ (unless otherwise specified)
$\Psi_0 \leftarrow I$
for $j \in [J]$ **do**
  $\Psi_j \leftarrow P^{2^{j-1}} - P^{2^j}$
  if SPARSE **then**
    Calculate $\tilde{\Psi}_j$ given Ψ_j, ϵ_1 per algorithm 1
  end if
end for
$\omega \leftarrow [1, 1...2, 2...2^J, 2^J]$ with as many 2^j's as columns in Ψ_j or $\tilde{\Psi}_j$
$\Phi \leftarrow \begin{bmatrix}\Psi_0 & \Psi_1.. & \Psi_J\end{bmatrix}$ or $\Phi = \begin{bmatrix}\tilde{\Psi}_0 & ... & \tilde{\Psi}_J\end{bmatrix}$ as appropriate.
Normalize columns of Φ
$x \leftarrow OMP(\Phi, \frac{s}{\hat{s}}, \epsilon_2)$ (we normalize s in case we'd like to compare different signals)
$\ell(s) \leftarrow \langle w, |x| \rangle$, where $|x|$ is the elementwise absolute value of x
return $\ell(s)$.

An alternative algorithm, for the reduced set, is to use $\tilde{\Psi}_j$ to calculae $\tilde{\Psi_{j+1}}$. This would go like so:

Algorithm 3 Calculating Smoothness (Alternative)

Input: A diffusion operator M and signal s defined on a set $\mathcal{X}$. SPARSE, a boolean variable encoding whether to use the full Ψ_j or $\tilde{\Psi}_j$, an ϵ to dictate $\tilde{\Psi}_j$. OMP a routine for matching pursuit.

Output: $\ell(s)$, a measure of localization of s on the set $\mathcal{X}$.

$P \leftarrow \frac{1}{2}(I + M)$
$J \leftarrow \log(|\mathcal{X}|)$ (unless otherwise specified)
$\Psi_0 \leftarrow I$
for $j \in [J]$ **do**
  $\Psi_j \leftarrow (I - P^{2^{j-1}})\tilde{\Psi_{j-1}}$
  Set $\tilde{\Psi}_j$ per our algorithm.
end for
$\omega \leftarrow [1, 1...2, 2...2^J, 2^J]$ with as many 2^j's as columns in $\tilde{\Psi}_j$
$\Phi \leftarrow \begin{bmatrix}\tilde{\Psi}_0 & ... & \tilde{\Psi}_J\end{bmatrix}$ as appropriate.
Normalize columns of Φ
$x \leftarrow OMP(\Phi, \frac{s}{\tilde{s}})$ (we normalize s in case we'd like to compare different signals)
$\ell(s) \leftarrow \langle w, |x| \rangle$, where $|x|$ is the elementwise absolute value of x
return $\ell(s)$.

A benefit to this version is that Ψ_j has as many rows as $\tilde{\Psi_{j-1}}$, so our work keeps decreasing, which becomes highly beneficial at larger and larger scales. Note it is more efficient to caalculate P via repeated squaring. This way, we calculate $P^{2^j} \leftarrow (P^{2^{j-1}})^2$ in time $\mathcal{O}(n^2)$. So for J scales, runtime is at worst $\mathcal{O}(n^2 \log(n))$. Compare this to an eigendecomposition, which could take time up to $\mathcal{O}(n^3)$. So for large n and relatively few scales, this is more effective.

Example on the Bunny Graph

As a start, let's use the redundant ϕ, which contains all wavelets at all scales. As a sanity check, we can consider graph signals of the form $s = M^t \delta_i$ and verify that, as we increase t, ℓ increases.

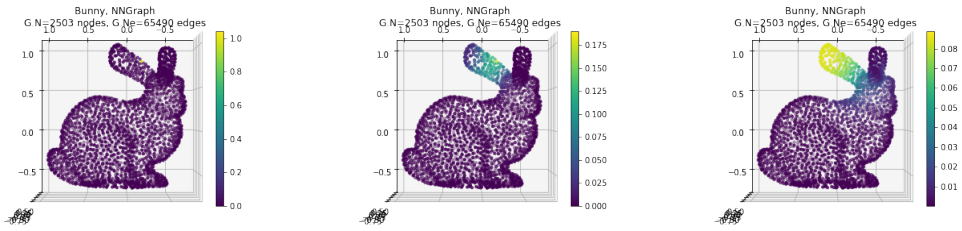

Figure 2: Here we have visualized signals of the form $M^t \delta_i$ with $t = 0, 2^3, 2^6$ and $i = 1400$. We see that as we increase t, the signal should intuitively be less "local." Accordingly, the respective measures of locality are approximately $50, 157$, and 242

We could also repeat this using our QR-reduced dictionaries. In this case, the signals are the same, but the estimates of locality are $24, 357$, and 492. So wee see that in the simplest case,

our algorithm does what it's supposed to.

Moving Signals Across the Sphere

Another thing we might imagine doing is fixing one signal on a sphere and moving another to see how the relative localization changes. This is visualized in the below diagram:

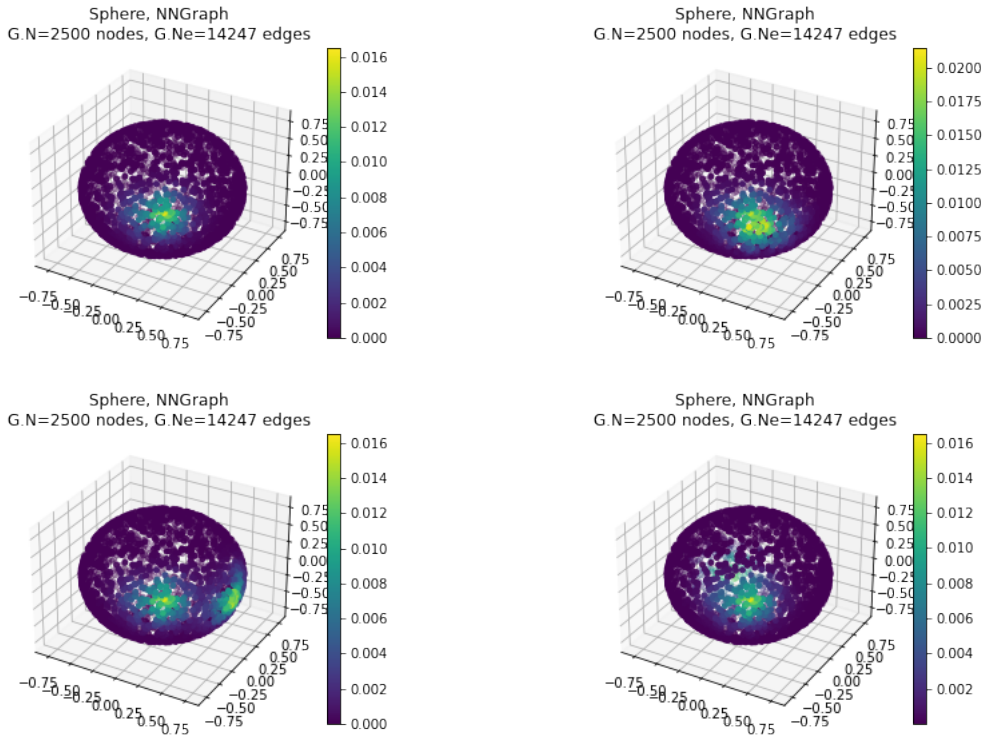

Figure 3: We can see that as we continue this process, we would hope that the signal is identified as less and less localized. But once the signal becomes more and more bimodal, the level of localization stabilizes. What we see is that the corresponding levels of localizations $684, 728, 741$, and 421. Here we use the sparser wavelets with $\epsilon = 0.001$

This example demonstrates that, at the bare minimum, the most centered signal indeed receives the lowest score. Then, the large jump in value from the first to the second signals that "wider" signals are weighed more heavily than bimodal ones. Whether this is desirable is questionable; maybe it is, maybe it isn't. If it is not, it may be worth revisiting the weight vector ω.

Non-Frequency Approach

Another possible approach is to avoid scanning the signal using frequencies altogether and use a quantity which captures the "spread" of a signal. Suppose we have our set $\mathcal{X}$ and, rather than a diffusion operator, some distance $d(\cdot,\cdot)$ defined on $\mathcal{X}\times\mathcal{X}$. In this case, if we have a nonnegative signal $s\in\mathbb{R}_+^n$, we can normalize s by taking $\hat{s}=\frac{s}{\sum_{x\in X}s(x)}$, and view $\hat{s}$ as a probability distribution. Now we can examine the quantity:

$$\mathcal{S}(s)=\sum_{x,y}\hat{s}(x)\hat{s}(y)d(x,y)$$

Notice that in the simplest case, if s is binary on $\mathcal{X}$ (ie $s\in\{0,1\}^{|\mathcal{X}|}$), we couls say $s=1$ over a set $A\subset X$ and observe

$$\mathcal{S}(s)=\frac{1}{|A|^2}\sum_{x,y\in A^2}d(x,y)$$

Which can be viewed as average distance between vertices where s is "on." So then $\mathcal{S}(s)$ for general s serves as a continuous extension of this. Similarly, if we think of $\hat{s}$ as a distribution, then, the joint distribution of (x,y) is simply $\hat{s}(x)\hat{s}(y)$. So then we have the interpretation,

$$\mathcal{S}(s)=\sum_{x,y\in\mathcal{X}\times\mathcal{X}}\mathbb{P}\{x,y\}d(x,y)=\mathbb{E}_{x,y\sim\hat{s}}[d(x,y)]$$

Note that if $\mathcal{X}$ is finite, then we can store $d(\cdot,\cdot)$ in a matrix D. So then,

$$\mathcal{S}(s)=\frac{s^TDs}{s^T\mathbf{1}\mathbf{1}^Ts}$$

A possible normalization factor, for the sake of comparison, could be

$$C(s):=(\sum_x s(x))^2-\sum_x s(x)^2=\hat{s}^T(J-I)\hat{s}=\mathbb{E}_{x,y\sim\hat{s}}(1-\delta(x,y))$$

Which is the expected distance over the complete graph with weight $\omega:E\to\mathbb{R}$ with $\omega(x,y)=\delta(x,y)$. For diracs s, $C=0$, this sends the normalized measure to ∞, which could be a nice way to weed out trivially localized signals.

So in practice, we could choose $d(\cdot,\cdot)$ in any number of ways. If $\mathcal{X}\in\mathbb{R}^n$, we could simply use the Euclidean distance between data points and store them in a matrix D. If X is not spatial, d could be the length of the shortest path. We could also let $d(x,y)$ be, for example, the diffusion distance between x and y, given by $\|P(x,\cdot)-P(y,\cdot)\|^2_{L^2(X,dP/d\pi)}$ where $P:\mathcal{X}\times\mathcal{X}$ is a probability kernel with stable distribution π. As shown by Coifman & Lafon, this would be equivalent to $\|\Psi(x,\cdot)-\Psi(y,\cdot)\|^2$ where Ψ denotes the matrix whose columns are right eigenvectors of P. Another possible choice could be the distance used by PHATE, given by $\sqrt{\|\log P^t(x,\cdot)-\log P^t(y,\cdot)\|}$ (t is by default the Von Neumann entropy).

Thus, we present the following algorithm:

Algorithm 4 Algorithm For Computing Locality 2

Input: a graph G with associated affinity or adjacency matrix A. Set of signals $S = \{s_1...s_m\}$ on the graph

Output: A set of localities $\{l_1...l_m\}$ for each signal.

Calculate the diffusion operator $P = AD^{-1}$, where D is the matrix of row sums of A.

Calculate the Von Neumann entropy t of P.

Calculate $\log(P^t)$. Optionally, run PCA on this matrix to expedite the next step.

Form a matrix D such that $D_{i,j} = \sqrt{\| \log P^t(i,\cdot) - P^t(j,\cdot)\|}$

for $i \in 1..m$ **do**

$s_i \leftarrow \frac{1}{\sum_x s(x)} s^t D s$

$c_i \leftarrow \frac{1}{\sum_x s(x)} s^t (J - I) s$

$l_i = \frac{s_i}{c_i}$

end for

return $l_1...l_m$.

Remark: For large graphs G, D is highly expensive to compute, requiring $\mathcal{O}(n^3)$ arithemetic operations. Instead, you could generate the matrix $\log(P)$ in time $\mathcal{O}(n^2)$, reduce to dimensionality d via PCA (like $d = 100$) and then calculate D in time $\mathcal{O}(n^2 d)$.

Results

Bunny Graph: On the same examples for the Bunny (Figure 2) as before, the corresponding measures of locality are 30, 89, and 160.

Moving Signals on the Sphere: A similar example to one on the sphere yielded 196, 197, 292, and 443 as we spread the signal to the other side. Thus, whereas the frequency based algorithm just cared about decomposing signals into sparse wavelets, this new meaasure *does* penalize multimodal distributions.

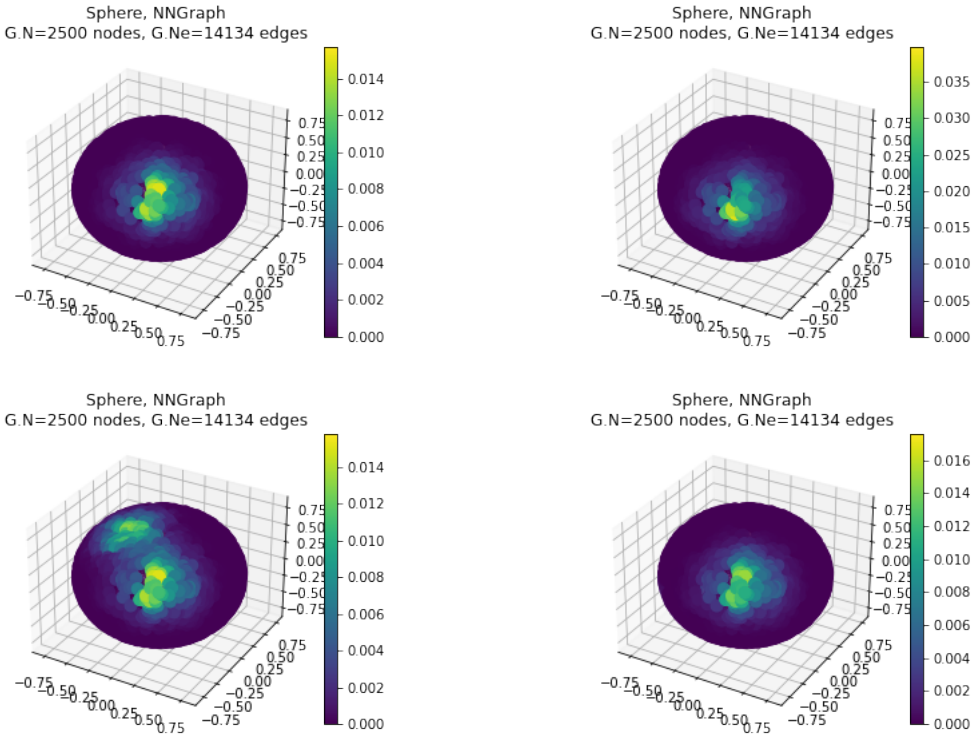

Figure 4: Note that in the final figure, there is a second mode on the other side of the sphere which is not visible. Again, the corresponding locality values were 197, 196, 292, and 443, which is intuitively what we might hope for! While signal 1 is given by heat diffusion centered at just 1 point and signal 2 is the sum of two diffusions (so the intent was admittedly for 1 to be more local), just from visual inspection, signal 2 is the more localized one.

On the Single Cell Data Now, let's see what the resulting output is on cell-cell data. Here, we look at a signal given by expression of 1991 genes. We also take an order 32 diffusion of the data to counteract dropout. Furthermore, we rank the *normalized* localities to obtain a list of the least to most "local" genes. For reference, here is the distribution of the data in general:

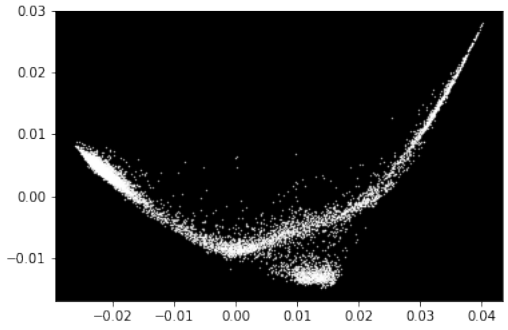

Figure 5: Here each point represents a data cell. Coordinates are given by a PHATE embedding using default settings. The diffusion operator is generated from Euclidean distance in a PCA-reduced feature space.

And here are genes 0, 500, 1000, and 1500 (numbered in increasing order of l):

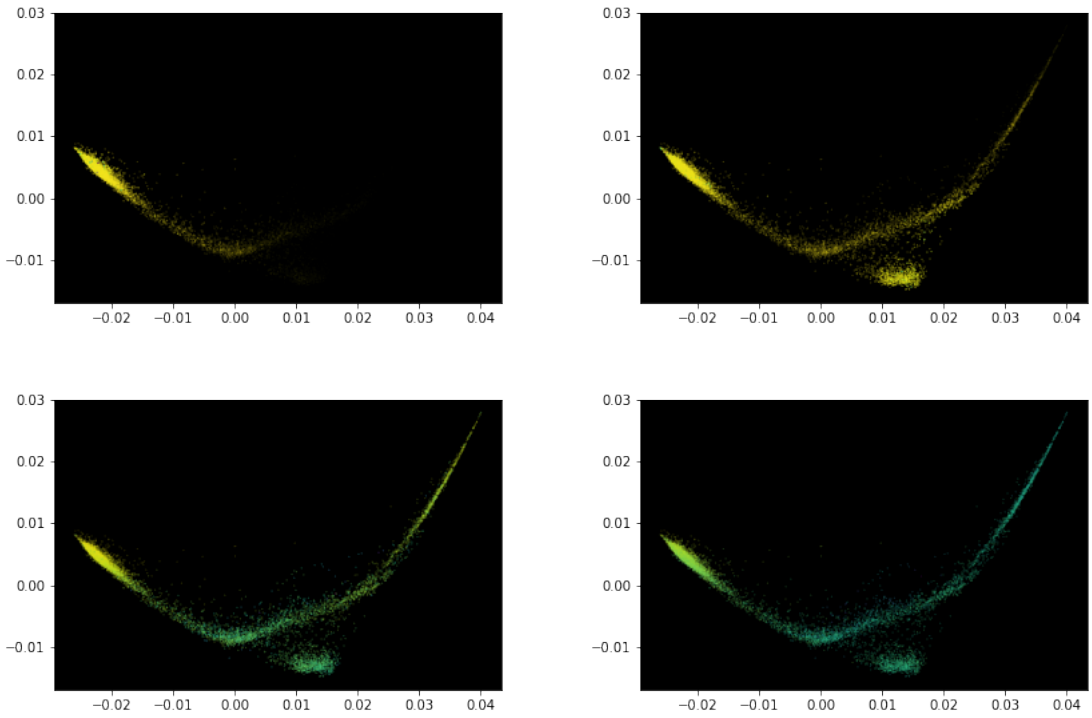

Figure 6: Here we have drawn plots where we use color and size to indicate gene intensity. The visualization is a 2-dimensional phate embedding (default settings). The corresponding locality measures are 110, 250, 285, and 306. Fortunately, these signals to intuitively seem to be more and more spread out.

Robustness to Dropout?: Another natural question is: how necessary was it to diffuse the signal to combat dropout? Certainly, if we're working with frequencies, this is necessary. But here our appraoch isn't so sensitive. Let's go ahead and re-rank the signals, but using their *raw* values over the genes. The result will be two vectors ℓ_r and ℓ_d, where $\ell_r(i)$ contains the localization of the raw signal of gene i and $\ell_d(i)$ is the diffused localization. In our case,

$$\ell_r = (306.0, 286.3, 179.9, 207.3...179.2, 287.5, 295.0)$$

$$\ell_d = (302.3, 283.4, 208.8, ...187.2, 287.4, 234.9)$$

By initial inspection, these values roughly aline, but we can also go ahead and compare how they compare on a *relative* basis. That is, let r_r be the indices of ℓ_r in increasing order and r_d likewise. We can then look at the Kendall-Tau distance between ℓ_r and ℓ_d. Unfortunately, here we have $\tau = 0.00672$ and a p-value of nearly 0.65. Even after comparing this to a more moderate scale of diffusion, we get weak results. This doesn't mean they're not correlated, just that perhaps that sensitivity to small perturbations affected the ranking too much to observe a

significant value here.

While it would have been nice to be able to say the rankings are nearly equivalent, we can also look at a correlation test. Here, we have 90% correlation with a p-value of 0 (up to machine precision). So there is consistency regardless of if we account for dropout.

Bayesian Approach

How are signals like s generated? One way we might imagine generating a localized signal on a data manifold $\mathcal{X}$ is to first pick some center $x^\star$. And then for all $x \in \mathcal{X}$, set

$$s(x) \propto k(x, x^\star)(1 + \epsilon_x)$$

Where k is some symmetric kernel $\mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $\{\epsilon\}_{x \in \mathcal{X}}$ are a sequence of iid errors. For simplicity, say $\epsilon \sim \mathcal{N}(0, \sigma)$ for some unknown σ. Thus, our signal generation process goes like so:

$$s(x) = \beta \cdot k(x, x^\star) \cdot Z_x$$

Where $Z_x \sim \mathcal{N}(0, 1)$. As an additional simplifying assumption, say k is a Gaussian kernel with bandwidth h. We can then find maximum likelihood estimates of $x^\star, h, \sigma$, and β. To do this, note we can write likelihood as:

$$\ell(s; x^\star, \beta, h, \sigma) = \prod_x p(\beta k_h(x, x^\star) Z_x = s(x)) = \prod_x p_Z(\frac{s(x)}{\beta k_h(x, x^\star)})$$

So if we then take the log likelihood:

$$= \sum_x -\log(\sigma) - \frac{1}{2}\log(2\pi) - \frac{1}{2\sigma^2}(1 - \frac{s(x)}{\beta k_h(x, x^\star)})^2$$

$$= \sum_x -\log(\sigma) - \frac{1}{2}\log(2\pi) - \frac{1}{2\sigma^2}(1 - \frac{s(x)}{\beta} e^{\frac{\|x - x^\star\|^2}{2h^2}})^2$$

Which is not a forgiving expression. The parameters could be estimated iteratively. Then, we could examine the likelihood of s given its MLEs.

Misc. Ideas:

- Finding geodesics on the graph $\Gamma := (\Omega, P(\Omega))$ with P the kernel density estimate — could estimate through MCMC. Hamiltonian MC / simulated annealing / Gibbs sampler on path segment?
- Find ways to reduce the dictionary even more or faster. Random QR decomposition? Maybe even just PCA here.
- DiffusionEMD to uniform
- Flux from "s" to "not s." How to define continuously?
- Markov Random Fields (Binarize & apply Hammersley-Clifford?) as HMM for dropout?

Appendix

Selecting $\tilde{\Psi}_j$ — Algorithm Correctness

Proposition: $\tilde{\Psi}_j$ as outputted by Algorithm 1 satisfies equation (1). *Proof:* This is equivalent to proving:

$$\|\Psi_j - \tilde{\Psi}_j(\tilde{\Psi}_j{}^T\tilde{\Psi}_j)^{-1}\tilde{\Psi_j}^T\Psi_j\|^2 \leq \epsilon\|\Psi_j\|^2 = Q \cdot R(p(1)...p(i))$$

Recall that our algorithm first takes a pivotal QR decomposition of Ψ_j. Our algorithm first finds i such that $\|\Psi_j\| - \sum_{k=0}^{i}\sum_{j=k}^{n} R_{k,j}^2 \leq \epsilon\|\Psi_j\|$. Or equivalently, so that $\|\sum_{k=0}^{i}\sum_{j=k}^{n} R_{k,j}^2\| \geq (1-\epsilon)\|\Psi_j\|$. First, observe that $\|\Psi_j\|^2 = \|QR\|^2 = \|R\|^2$ because Q is orthonormal.

Observe that for all $k \leq i$ that $R_{k,k} > 0$ per how the pivotal QR decomposition algorithm works. If any columns are linearly dependent, those will be attributed to indices higher than i. I also claim that. Thus, $C(\tilde{\Psi}_j) = C(q_1...q_i)$, where $q_1...q_i$ are the first i columns of Q. Thus, if we assemble these columns into a matrix Q_i, we find that the projection matrix onto $C(Q_i)$ is the same as the projection onto $\tilde{\Psi}_j$. Thus,

$$\|\Psi_j - \tilde{\Psi}_j(\tilde{\Psi}_j{}^T\tilde{\Psi}_j)^{-1}\tilde{\Psi_j}^T\Psi_j\|^2 = \|\Psi_j - Q_iQ_i^T\Psi_j\|^2$$

Suppose the columns of Ψ_j are $\psi_j^1...\psi_j^n$. We can also compute the above norm by column:

$$= \sum_k \|(I - Q_iQ_i^T)\psi_j^k\|^2$$

But since Q is orthonormal and ψ_j is in the span of Q, we can represent $(I - Q_iQ_i^T)$ by the matrix $\hat{Q}_i\hat{Q}_i{}^T$, where $\hat{Q}_i$ contains columns $i+1...n$ of Q. So:

$$= \sum_k \|\hat{Q}_i\hat{Q}_i{}^T\psi_j^k\|^2$$

Note that p is a permutation, so we can rewrite the order as:

$$= \sum_k \|\hat{Q}_i \hat{Q}_i^T \psi_j^{p(k)}\|^2$$

Of course, observe that by the way the QR decomposition works, column $p(k)$ of Ψ_j is:

$$\psi^{p(k)} = \sum_{l \leq k} q_l R_{l,k}$$

Plugging this in, our original norm is: r

$$= \sum_k \|\hat{Q}_i \hat{Q}_i^T \sum_{l \leq k} q_l R_{l,k}\|^2 = \sum_k \| \sum_{l \leq k} \hat{Q}_i \hat{Q}_i^T q_l R_{l,k}\|^2$$

Since the columns of q_l are orthonormal:

$$= \sum_k \| \sum_{l \leq k} \mathbf{1}\{l > i\} q_l R_{l,k}\|^2$$

Applying Parseval's identity:

$$= \sum_k \sum_{l \leq k} \mathbf{1}\{l > i\} R_{l,k}^2$$

Which is precisely equal to the *sums of squares* of rows $i+1...n$ of R. Thus,

$$\|\Psi_j - \tilde{\Psi}_j (\tilde{\Psi}_j^T \tilde{\Psi}_j)^{-1} \tilde{\Psi}_j^T \Psi_j\|^2 = \sum_{k>i} \sum_{l \geq k} R_{k,l}^2 = \|R\|^2 - \sum_{k \leq i} \sum_{l \geq k} R_{k,l}^2$$

By design,

$$= \|\Psi_j\|^2 - \sum_{k \leq i} \sum_{l \geq k} R_{k,l}^2 \leq \epsilon \|\Psi_j\|^2$$

Thus, the claim must be true as claimed $\square$.