TopoRNet

Analysis and Visualization of Topology Representing Networks

Josh Taylor

Contents

Preface

`TopoRNet` is an R package providing a C++ template for manipulation of and computation with a Topology Representing Network (or TRN, which is also introduced in this document). As TRNs represent mappings from an **input space** to an **output space** they possess an additional layer of complexity when compared to standard mathematical graph representations. While several R packages for graph analysis currently exist (the `igraph` package being the most popular) none (inherently) contain the machinery required to properly handle this additional complexity. `TopoRNet` addresses this shortcoming in the R community by providing the following functionality which, to our knowledge, no other package does:

- A templated C++ class to facilitate storage of TRNs and derivative products.
- Calculation of the **Topographic Product** and several **Topographic Functions.** Collectively, these Topology Preserving Measures (TPMs) provide a metric for assessing the *degree* to which topology is preserved, when it is represented by the TRN.
- The **CONNvis** visualization (and calculation of its required statistics) which aides human and automated cluster inference from TRNs.
- Intelligent sparsification of a TRN (which improves the quality of cluster extraction) via its CONNvis statistics and the recently introduced `DM-Prune` methodology.

- C++ implementations of the above (based on Rcpp and RcppArmadillo)
- Optional parallel computation of the above (as applicable, via RcppParallel)

While a TRN can arise from several sources, it is most commonly a product of unsupervised manifold learning via the Self-Organizing Map. As such, the R package `SOMDisco` will also be utilized in this documentation. These packages are intended to work together to facilitate SOM-based cluster discovery. A 20x20 hexagonal SOM which has learned the 100-dimensional, 20-class cov500-4000 `SHGRWalk` hyperspectral image cube (more information here) has been included with `TopoRNet` for example purposes in the variable `SHGR.SOMList`. The dimension-wise inner 95% quantile range of each of the 20 data classes (denoted by letters "A" - "T") is given below:

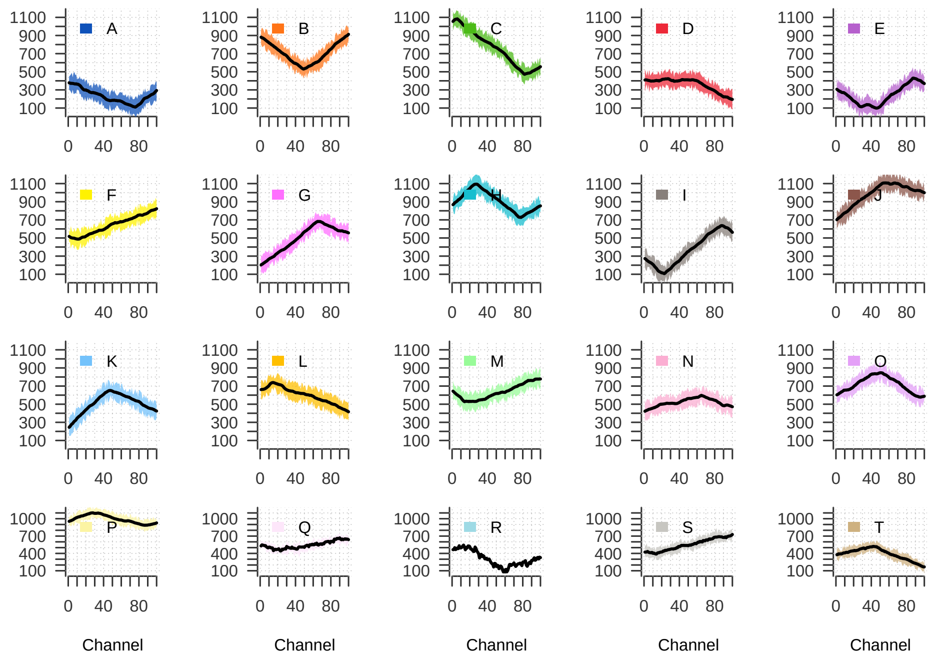


Figure 1: SHGRWalk 20-class Class-wise Statistics

The learned SHGR SOM shows good organization when the training data labels are propagated to each neuron by the SOM mapping:

```
## Load the learned SOM included with TopoRNet
SHGR.som = SOMDisco::SOM$new()
SHGR.som$load_list(TopoRNet::SHGR.SOMList)
% ++ calculating lattice (x,y) coordinates ... done
% ++ calculating neuron lattice adjacencies ... done
% ++ calculating geodesic lattice distances between neurons ... done
% ++ assigning geodesic lattice distances to distlist ... done
% ++ calculating lattice tile vertices ... done
% Storing the annealing schedule as:
% t          alpha         beta         gamma         sigma
% 16384          0.5         0.05         0.005         3
% 81920          0.25         0.025         0.0025         2
% 163840          0.1         0.01         0.001         1
% 409600          0.05         0.005         0.0005         1
% 1638400          0.01         0.001         0.0001         1
## Visualize it
SOMDisco::vis_som_label(SOM=SHGR.som, text.cex = 0.8, text.font = 2, rprop = 0)
```

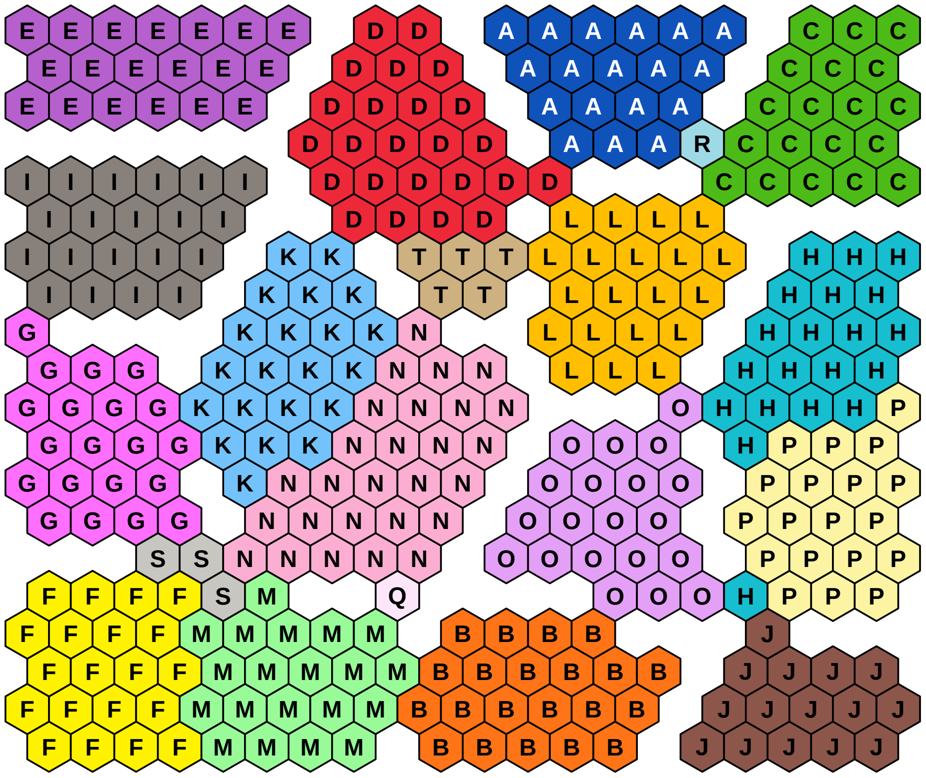


Figure 2: Learned SHGRWalk SOM Lattice

1 Installation

`TopoRNet` is available from github (not CRAN), installable via

```r
devtools::install_github("somdisco/TopoRNet")
```

The `DESCRIPTION` file in the package source lists the following 3rd-party R package dependencies:

```
Imports:
    Rcpp (>= 1.0.1),
    igraph,
    dplyr,
    Rdpack,
    ggplot2,
    changepoint,
    ggrepel
LinkingTo: Rcpp, RcppArmadillo, RcppParallel
```

Additionally, the packages `SOMDisco` and `VorVQ` are helpful for generating the various inputs required for the calculations exposed in `TopoRNet`. TRN storage and manipulation is made efficient through use of object-oriented C++ behind the scenes. For those who may desire custom or expanded functionality, the class template is implemented in header-only fashion in the `inst/include` directory of the package source.

2 Background

A **T**opology **R**epresenting **N**etwork (or **TRN**, [1]) arises from the quantization of a set of training data $X = \{x_s \in \mathbb{R}^d\}_{s=1}^{nX}$ by a set of n_W codebook or **prototype** vectors

$$W = \{w_j \in \mathbb{R}^d\}_{j=1}^{n_W}.$$

The TRN is the graph whose vertices are the prototypes or, equivalently, their indexing set $N = \{1, 2, \dots, n_W\}$, and whose edges represent the **topological adjacencies** of the $\{w_j\} \in \mathbb{R}^d$. $\mathbb{R}^d$ is known as the TRN's **input space**, while the indexing set N is known as its **output space**; the vector quantizer underlying the TRN maps $\mathbb{R}^d \to N$. The most well known TRN is the Delaunay Graph $\mathcal{D}$ [2] which is induced by the Voronoi tessellation [3] of the set of prototypes W. The vertices of $\mathcal{D}$ are the set N; an edge connects $j \leftrightarrow k \in N$ IFF the Voronoi cells V_j and V_k, which are polytopes in $\mathbb{R}^d$ generated by w_j and w_k, respectively, intersect (i.e., share a face $\in \mathbb{R}^{d-1}$). Other TRNs with various properties are formed as sub-graphs of $\mathcal{D}$, such as the Gabriel Graph [4] and Relative Neighbourhood Graph [5].

Martinetz & Schulten recognized that, for TRNs whose prototypes have been formed from sample data X to represent an unknown manifold $\mathcal{M} \subset \mathbb{R}^d$, the TRN that most faithfully represents $\mathcal{M}$ is another Delaunay sub-graph they call the **Masked Delaunay Graph** $\mathcal{D}^{\mathcal{M}}$. The "masking" procedure intersects $\mathcal{M}$ (via its representation by W) with $\mathcal{D}$ and retains only those edges in $\mathcal{D}$ which represent prototypes that are *adjacent on* $\mathcal{M}$. By explicitly considering $\mathcal{M}$, the adjacencies in $\mathcal{D}^{\mathcal{M}}$ are capable of representing complex manifolds (particularly, completely embedded structure in $\mathbb{R}^d$).

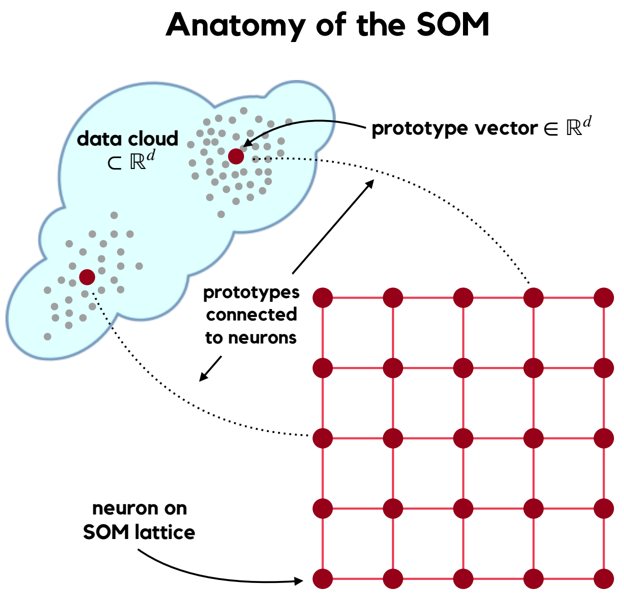
Taşdemir and Merényi [6] extend $\mathcal{D}^{\mathcal{M}}$ by weighting its edges by the number of samples which have informed the interpretation of connectivity on $\mathcal{M}$. This graph, known as **CADJ**, exposes not only the *existence* of topological connectivities but also their *strengths*. The weight of edge $i - j$ is given by

$$CADJ_{ij} = \#\{x : BMU1(x) = i \,\&\, BMU2(x) = j\}.$$

where $x \in X$ and $BMU1(x)$ and $BMU2(x)$ return the index of the first and second **B**est **M**atching **U**nit to x in W, respectively. The above expresson specifies each element of the $(n_W \times n_W)$ weighted adjacency matrix of the CADJ graph. As defined, the weights $CADJ_{ij}$ are the counts of data which fall into the second-order Voronoi cells [3] V_{ij} of the second-order Voronoi tessellation generated by W.

Analysis of the CADJ graph is useful for structural inference from learned representations of manifolds. More densely connected regions in $\mathbb{R}^d$ (i.e., edges with higher CADJ adjacency weights) are more likely to indicate cohesive structure; conversely, weak topological connectivities can signal either “seams” in $\mathcal{M}$ (areas where structure changes), or possibly very subtle structural differences that are easily overlooked by considering topography alone (i.e., by considering only the distances between adjacency prototypes).

In some special cases the TRN’s output space (prototype indexing set N) possesses its *own topology.* Kohonen’s Self-Organizing Map [7] is probably the most famous example of such a case. Relying on competitive neural learning, SOM training simultaneously forms the prototypes $W \in \mathbb{R}^d$ and a (self-organized) representation of these prototypes as neurons on rigid lattice structure $\mathcal{L}$ (the SOM’s output space, typically a rectangular or hexagonal lattice in $\mathbb{R}^2$ or $\mathbb{R}^3$).


Anatomy of the SOM
data cloud ⊂ $\mathbb{R}^d$
prototype vector ∈ $\mathbb{R}^d$
prototypes connected to neurons
neuron on SOM lattice

The visualizations afforded by the SOM’s output space have (for decades now) been harnessed for various tasks in pattern recognition and clustering. In the context of topology representation it has additional implications, such as allowing measures of the quality of this representation and, most importantly, providing the scaffolding required to view the TRN for data of arbitrary dimension. Without such a visualization the TRN is without much practical use, as it is “representing” topology on the indexing set N which has no natural “home” (no space in which we, as analysts, can view or interact with it).

Below we initialize a `TRN` object (which is the container for all information required to represent a TRN) for the learned SHGR SOM via the exposed class constructor in `TopoRNet`. The `initialize_TRN` method takes the topology representations of both the SOM's input space (CADJ) and output space (the adjacency matrix of neurons prescribed by the lattice topology):

```
## Build a new TRN object for SHGR
SHGR.trn = TopoRNet::TRN$new()
SHGR.trn$initialize_TRN(SHGR.som$CADJ, SHGR.som$nu_ADJ)
% Initializing TRN object:
% ----------------------------------------------------------------------
% Storing Output Topology:
% ++ edge list ... done
%    2710 edges stored
% ++ vertex degrees ...  done
% ++ geodesic distances ...  done
%    max dist = 29
% ++ neighborhood sizes ... done
% ----------------------------------------------------------------------
% Storing CADJ Topology:
% ++ edge list ... done
%    1874 edges stored
% ++ weights ... done
% ++ vertex degrees ...  done
% ++ forward folding lengths ... done
% ++ backward folding lengths ... done
% ++ topology preserving radius ... done
%    CADJ_TP_radius = 2
% ++ activating all edges ... done
% ----------------------------------------------------------------------
% Calculating CADJvis statistics
% ++ CADJ local rank stats ... done
% ++ CADJ global rank stats ... done
% ++ CADJ length stats ... done
% Call $get_CADJvis_stats to view
% ----------------------------------------------------------------------
% Storing CONN Topology:
% ++ edge list ... done
%    2234 edges stored
% ++ weights ... done
% ++ vertex degrees ...  done
% ++ forward folding lengths ... done
% ++ backward folding lengths ... done
% ++ topology preserving radius ... done
%    CONN_TP_radius = 3
% ++ activating all edges ... done
% ----------------------------------------------------------------------
% Calculating CONNvis statistics
% ++ CONN local rank stats ... done
% ++ CONN global rank stats ... done
% ++ CONN length stats ... done
% Call $get_CONNvis_stats to view
% -----------------------------------------------------------------
```

As visible in the output messages, `initialize_TRN` computes the CADJvis and CONNvis statistics of the TRN which are covered in the next chapter. The `TRN` object stores the graphs as an edge list, which is a two-column matrix indicating the (i, j) indices of the vertices connected by each edge, and an edge weight vector whose elements list the weights of edges defined in the edge list. The edge list and their corresponding weights which are stored during initialization can be accessed via:

```
## First few CONN edges and their weights
head(SHGR.trn$CONN_EL)
%      [,1] [,2]
% [1,]    2    1
% [2,]    3    1
% [3,]   21    1
% [4,]   22    1
% [5,]   83    1
% [6,]   84    1
head(c(SHGR.trn$CONN))
% [1] 57  2 69  3  2  1

## First few CADJ edges and their weights
head(SHGR.trn$CADJ_EL)
%      [,1] [,2]
% [1,]    2    1
% [2,]    3    1
% [3,]   21    1
% [4,]   22    1
% [5,]   83    1
% [6,]   84    1
head(c(SHGR.trn$CADJ))
% [1] 25  2 28  1  1  1
```

3 Visualizations

3.1 TopoView

As the dimension of a TRN's input space grows it must represent a (potentially) increasing number of adjacencies in $\mathbb{R}^d$ due to the explosion in the number of faces of the Voronoi cells underlying its representation. While the Delaunay graph is much more susceptible to this curse of dimensionality (because it ignores part of the information learned by the SOM), the masking process which produces CADJ makes it a very sparse Delaunay subgraph. This sparsity alone renders CADJ a much more useful tool for cluster discovery from SOMs, as is visible where the Delaunay graph of the SHGR data (left, computed in the package `VorVQ` whose partial output is included in the `SHGR.VORList` variable shipped with `TopoRNet`) is displayed alongside its corresponding learned CONN graph (right, CONN is the symmetrized version of CADJ: $CONN = CADJ + CADJ^T$) on the 20x20 lattice.

```
## Visualize the Delaunay graph connecting SOM neurons
SOMDisco::vis_som_TopoView(SOM = SHGR.som,
                           ADJ = TopoRNet::SHGR.VORList$DADJ, edge.lwd_range = c(1,1))

## Visualize the CONN graph connecting SOM neurons
SOMDisco::vis_som_TopoView(SOM = SHGR.som,
                           ADJ = SHGR.som$CONN(), edge.lwd_range = c(1,1))
```

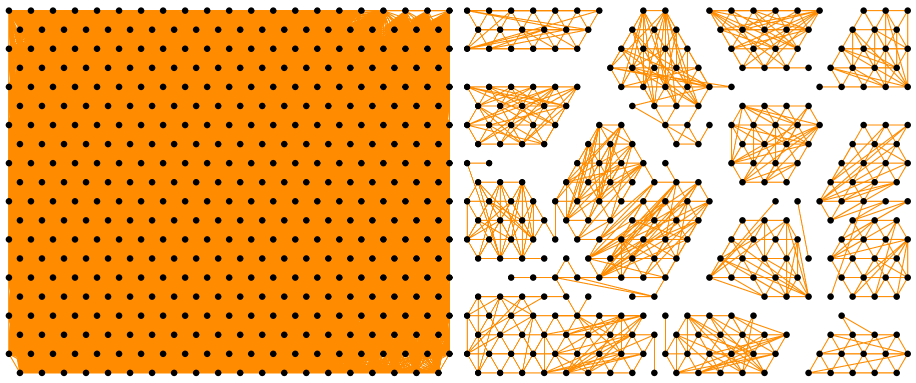
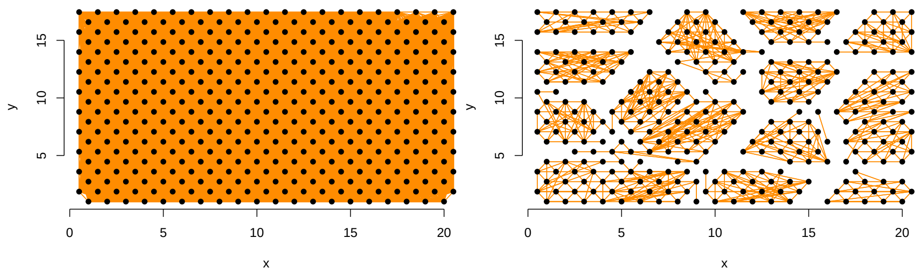

(a) Delaunay (b) CONN

Figure 3: SHGR SOM TopoView

The visualizations above are known as TopoView [8] and were created with the function `vis_som_TopoView` from the `SOMDisco` package, which is a wrapper to the function `vis_TopoView` from `TopoRNet` tailored for viewing graphs on a SOM lattice. We could, instead, have called `vis_TopoView` directly by manually specifying the lattice neuron coordinates from the SHGR SOM, but in this case the user will likely need to pre-specify R's `par()` options, e.g., aspect ratio or plotting limits, to produce a satisfactory figure (this is done automatically for lattices inside the `SOMDisco` wrapper):

```
## Visualize the Delaunay graph connecting SOM neurons
TopoRNet::vis_TopoView(ADJ = TopoRNet::SHGR.VORList$DADJ,
                       vertex.xy = SHGR.som$nu_xy, edge.lwd_range = c(1,1))
```

```
## Visualize the CONN graph connecting SOM neurons
TopoRNet::vis_TopoView(ADJ = SHGR.som$CONN(),
                            vertex.xy = SHGR.som$nu_xy, edge.lwd_range = c(1,1))
```



(a) Delaunay (b) CONN

Figure 4: SHGR SOM TopoView

3.2 CONNvis

TopoView is intended primarily for display of un-weighted and un-directed TRN's (like the Delaunay graph) on the SOM lattice. Recall, however, that CADJ and CONN are weighted TRNs, and additionally CADJ is a directed graph. The CONNvis visualization [6] was developed to incorporate this additional information in the form of visual indicators (edge colors and widths) of local and global statistical summaries of the edge weights. The result is a graph visualization that is much more amenable to interpretation and cluster discovery.

Briefly, edge colors represent the *local* importance (ranking) of edge weights, relative to the immediate neighbors of their source neurons (with red, blue, green, yellow, and grayscale denoting decreasing order of local importance). The CADJ / CONN local ranks are computed during TRN object initialization and stored in the fields

```
## CONN local ranks of first few graph edges
head(c(SHGR.trn$CONN_lrank))
% [1] 1 5 1 6 4 6

## CADJ local ranks of first few graph edges
head(c(SHGR.trn$CADJ_lrank))
% [1] 1 5 1 5 4 6
```

The mapping from local rank to color is also available:

```
## CONNvis colors of first few graph edges
head(c(SHGR.trn$get_CONNvis_colors()))
```

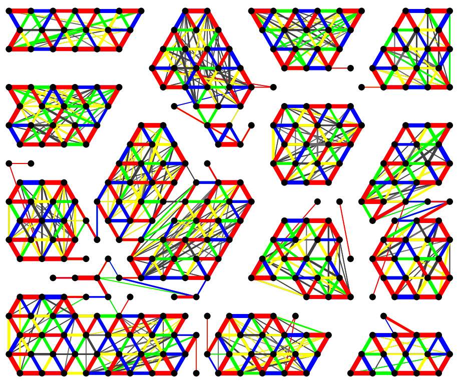
```
% [1] "red"    "grey25" "red"    "grey42" "yellow" "grey42"

## CADJvis colors of first few graph edges
head(c(SHGR.trn$get_CADJvis_colors()))
% [1] "red"    "grey25" "red"    "grey25" "yellow" "grey50"
```

Line widths are produced by binning each edge's weight relative to the entire (or global) set of edge weights, with bin boundaries selected automatically as the means of edge weights within each local rank. Binning with these endpoints assigns a global rank to each edge (e.g., an edge with global rank 1 means its weight is > mean(edges with local rank = 1)). The global ranks of each edge, along with their corresponding widths, can be accessed via:

```
## CONN global ranks and widths of first few graph edges
head(c(SHGR.trn$CONN_grank))
% [1] 1 6 1 6 6 7
head(c(SHGR.trn$get_CONNvis_widths()))
% [1] 7 2 7 2 2 1

## CADJ local ranks of first few graph edges
head(c(SHGR.trn$CADJ_grank))
% [1] 1 5 1 6 6 6
head(c(SHGR.trn$get_CADJvis_widths()))
% [1] 6 2 6 1 1 1
```

The combination of various colorings and widths of the visualized edges greatly improves the interpretation of manifold structure from a learned SOM, as visible below where the CONNvis (left) and CADJvis (right, which follows the same logic for edge color and width but highlights the asymmetry inherent in CADJ by only plotting half-lines) of the SHGR SOM have already separated neurons belonging to many of the 20 inherent clusters into visible subcommunities outlined by thicker red and blue connections. Details of the CONNvis procedure can be found in [6].

```
## CONNvis
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn)

## CADJvis
SOMDisco::vis_som_CADJvis(SOM = SHGR.som, TRN = SHGR.trn)
```


(a) CONNvis

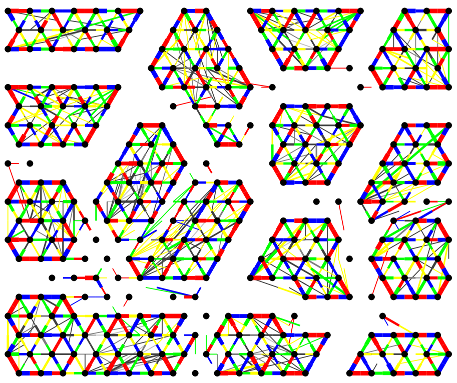
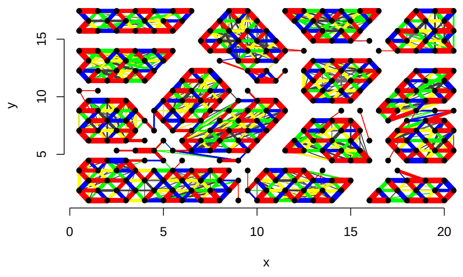

(b) CADJvis

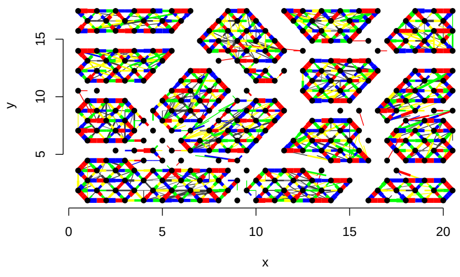
Figure 5: Weighted Topology Representations

Again, the above visualizations were rendered on the SOM lattice via wrapper functions exposed in the `SOMDisco` package, but we can re-create the base information in the figures by calling the corresponding lower-level visualization functions directly from `TopoRNet`:

```
## CONNvis
TopoRNet::vis_CONNvis(TRN = SHGR.trn, vertex.xy = SHGR.som$nu_xy)

## CADJvis
TopoRNet::vis_CADJvis(TRN = SHGR.trn, vertex.xy = SHGR.som$nu_xy)
```



(a) CONNvis



(b) CADJvis

Figure 6: Weighted Topology Representations

4 Pruning

Ideally, data structure (i.e., clusters) can be readily discerned from the CONNvis (or CADJvis) of a learned SOM via identification of its closed and connected (graph) communities. In practice, particularly for noisy data, CONNvis does not immediately indicate cleanly separated closed communities. This situation is depicted below, where the SHGR CONNvis has been visualized atop its "labeled lattice." Lattice labeling results from propagating the known (true) data labels to the lattice via the SOM mapping; here, the 20 distinct colors painted in each neural cell represent the 20 distinct truth labels of the SHGR image cube.

```r
## Visualize the SOM labels
SOMDisco::vis_som_label(SOM=SHGR.som,
                              text.cex = 0.8, text.font = 2, rprop = 0)

## Add CONNvis to the labeled SOM
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
```

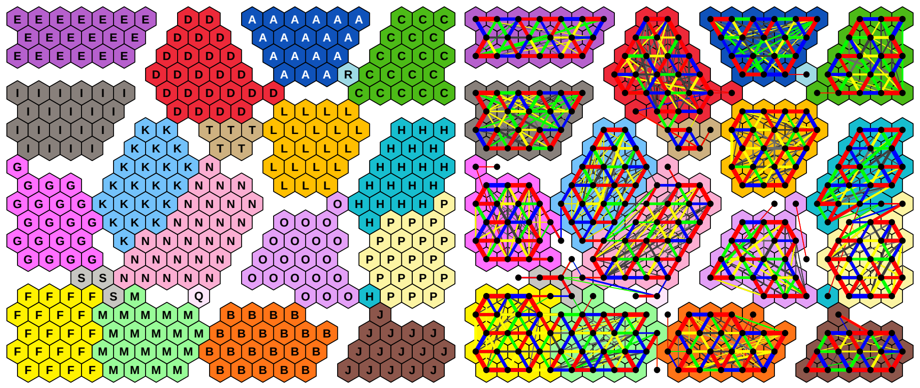


Figure 7: CONNvis Communities vs. True Clusters

While the truth labeling highlights the organization of the SHGR SOM, CONNvis is not completely disconnected along every cluster boundary visible on the lattice: notably, the clusters F & M, K & N, H & P, D & T, and A & R are connected by the CONN graph. In real unsupervised learning settings we do not have the truth labels to propagate to the SOM for inspection; in such situations relying on the CONNvis, as shown, can lead to improper cluster inference.

4.1 Grid Pruning

The obvious solution is to enforce further sparsity to CADJ or CONN (a process known as *pruning*) to help facilitate inference of its community structure. Because pruning CADJ/CONN introduces the possibility of destroying real clusters of neurons, with the introduction of CONNvis Taşdemir and Merényi introduced sensible methodology to guide the pruning process along a grid defined by the sets of unique local and global edge ranks. Intuitively, edges of low local or global rank are prime candidates for pruning. What constitutes "low" changes, of course, with every dataset. A conservative approach to defining "low rank" in a particular setting is formed by studying summary statistics of CADJ/CONN weights, grouped by rank. Ranks which primarily contain low-weight edges can usually be safely removed from the graph without deleterious impact on prominent cluster structure. `TopoRNet` provides built-in visualization functions for viewing these grouped statistics:

```
## CONNvis Stats by local rank
TopoRNet::vis_CONNvis_stats(TRN=SHGR.trn, group_by='lrank')

## CONNvis Stats by global rank
TopoRNet::vis_CONNvis_stats(TRN=SHGR.trn, group_by='grank')
```

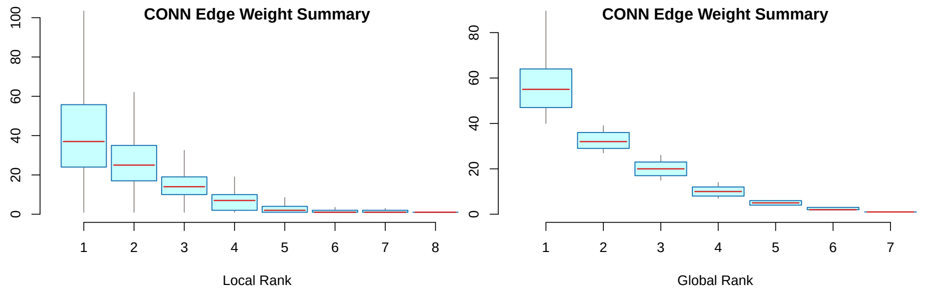


Figure 8: CADJ/CONN Graph Statistics

The above displays the CONN edge weight summary statistics (median + IQR + whiskers) grouped by the unique local and global edge ranks found in the graph. As the series of boxplots show, edges with both local and global rank > 5 have reliably low weights, suggesting their removal would not dramatically impact cluster inference from the SHGR CONNvis. A `TRN` object has many built-in tools to remove edges from either the CONN or CADJ graph (see the help documentation for any of the `prune_CADJ_*` or `prune_CONN_*` methods). Here, we prune CONN edges whose local and global rank is > 5:

```
## Prune CONN local ranks > 5
SHGR.trn$prune_CONN_lrank(5)
% Set 440 CONN edges inactive.

## Prune CONN global ranks > 5
SHGR.trn$prune_CONN_grank(5)
% Set 866 CONN edges inactive.
```

Note that the status messages from pruning indicate the number of edges that were inactivated by the method call. Edges are never removed entirely from the `TRN` object; instead, they are set as "inactive". At any point we can view the status of an edge, i.e., whether it is active (=1), or inactive (=0), via:

```
## Status of first few CONN edges
head(c(SHGR.trn$CONN_active))
% [1] 1 0 1 0 0 0
```

For example, the edge list of the first few edges inactivated by the above pruning can be retrieved via:

```
head(SHGR.trn$CONN_EL[SHGR.trn$CONN_active==0, ])
%      [,1] [,2]
% [1,]    3    1
% [2,]   22    1
% [3,]   83    1
% [4,]   84    1
% [5,]    1    3
% [6,]   45    3
```

Any previous pruning can be reversed by restoring all edge statuses to active:

```
## Un-do the above CONN pruning
SHGR.trn$restore_CONN_edges()

## Check that all edges are active
any(SHGR.trn$CONN_active == 0)
% [1] FALSE
```

The CONNvis (CADJvis) visualization functions exposed by `TopoRNet` only show active edges. Compared to the original CONNvis, we can see that pruning according to the above criteria, guided by the CONNvis group statistics, produces a graph which retains most of its important structure but is noticeably cleaner:

```
## Original CONNvis, with underlying labels
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)

## Re-prune according to the above, and visualize
SHGR.trn$prune_CONN_lrank(5)
% Set 440 CONN edges inactive.
SHGR.trn$prune_CONN_grank(5)
% Set 866 CONN edges inactive.

SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
```

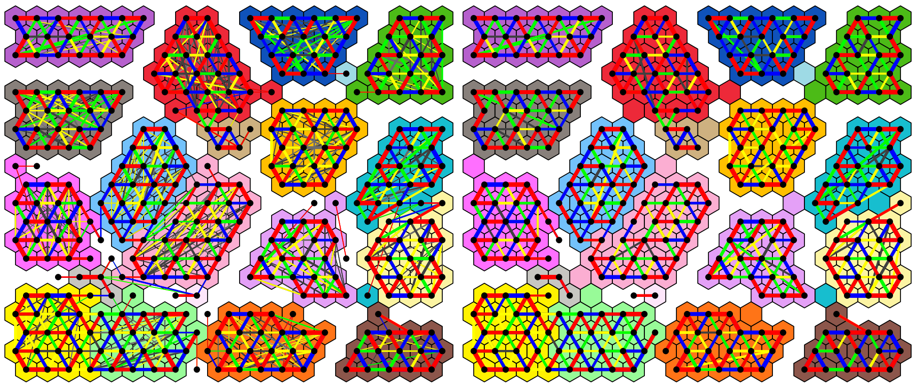


Figure 9: CONNvis Pruning Comparison

4.2 DM-Prune

Grid pruning, discussed above, is relatively simple and intuitive but lacks any mechanism to suggest optimal pruning levels. For example, clusters F & M, K & N and D & T were not cleanly separated by the grid pruning threshold levels selected above; more aggressive grid pruning might result in cleaner separation, but there is no barometer of how aggressive one should be in each particular data setting. As such, several grid pruning levels are typically specified (based on analysis of, e.g., the above rank statistics) and the resulting pruned CONNvis assessed for clarity. The DM-Prune procedure ([9], [10]) was introduced to facilitate the process of selecting optimal pruning levels via more formal statistical modeling.

DM-Prune is designed for the CADJ graph (not CONN) and comprises two steps. The first assigns a "quality" score μ to each CADJ edge following the same logic that human analysts use when assessing edge quality. μ is defined as the geometric mean of the local and global strength of each edge, a factor representing the edge's length on the SOM lattice (longer connections, resulting from forward topology violtions of the SOM mapping, are considered less reliable), and an optional fourth component representing edge reliability (derived from bootstrapped re-sampling of the CADJ graph). Each component contributing to the mean is normalized to $[0, 1]$, with the convention that $\mu \to 0$ signals lower edge quality. Edges are then ranked by their (increasing) μ score, with the unique set of ranks defining an (ordered) set of pruning steps. At each pruning step t, edges whose μ-rank is $< t$ are removed from the graph.

A Dirichlet-Multinomial (DM) likelihood of the sparse set of CADJ edge weights monitors the effects of pruning over "time" t. Recall from Section 2 that the edge weights $CADJ_{ij}$ are counts of sample data which are contained in the second-order Voronoi cell V_{ij} generated by the set of learned prototypes W. As a set of counts in each (highly irregular) second-order Voronoi "bin", the Multinomial likelihood is the natural statistical model for CADJ. When combined with a

Dirichlet prior on the bin probabilities, the Dirichlet-Multinomial likelihood, re-computed at the increasingly sparse set of CADJ counts resulting from pruning at each step t, provides a formal model of the impact of removing a particular set of edges from the CADJ graph.

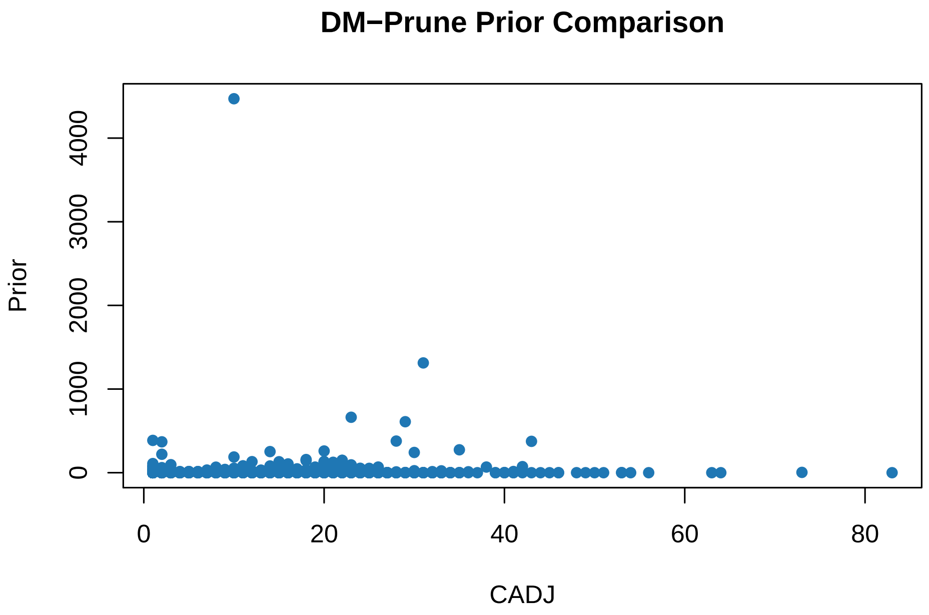
Moreover, specific Dirichlet priors allow an analyst the flexibility to incorporate new types of information into the modeling procedure. For instance, the Uniform prior (i.e., when all Dirichlet pseudocounts = 1) is commonly used to indicate uninformed prior beliefs. In this setting, where each Multinomial bin V_{ij} is a geometric object in $\mathbb{R}^d$, a more natural "uninformed" prior belief of each bin's count is proportional to its volume (i.e., for completely unstructured data, larger bins should intuitively have higher counts). The DM-Prune methodology recommends use of this latter volume-based prior to incorporate more of the manifold knowledge uncovered during SOM learning. A plot of the DM-likelihood calculated at each pruning step t (under a user-specified prior) is known as the DM-Prune Λ-path.

The companion R package `VorVQ` provides a way of estimating these Voronoi volumes via ellipsoid approximations to each second-order Voronoi bin. The `SHGR.VORList` variable included with `TopoRNet` contains the (partial) results of this computation for the SHGR SOM, which we use to construct a prior for CADJ below, in the form of an adjacency matrix whose dimensions match CADJ. The prior adjacency should be positive everywhere that CADJ is positive.

```
## Initialize an empty adjacency matrix
## with the same dimensions as CADJ
priorCADJ = 0*SHGR.som$CADJ

## Populate the entries corresponding to non-zero CADJ
## values with the pre-computed volumes of the corresponding
## second-order Voronoi Maximum Volume Inscribed Ellipsoids (MVIEs).
## The vor2_MVIE_volratio field of a VOR object stores these volumes,
## normalized to sum to 1.
priorCADJ[TopoRNet::SHGR.VORList$vor2_active] =
  TopoRNet::SHGR.VORList$vor2_MVIE_volratio

## Check that the prior is positive everywhere that CADJ is positive
any(c(priorCADJ[SHGR.som$CADJ > 0]) <= 0)
% [1] FALSE

## Check that the volume ratios sum to 1
sum(priorCADJ)
% [1] 1

## Re-normalize, such that the prior pseudo-counts
## equal the sum of CADJ weights
priorCADJ = priorCADJ * sum(SHGR.som$CADJ)
```

Once constructed, the CADJ prior (in adjacency-matrix format) is used to calculate the DMP-Prune Λ-path of a `TRN` object:

```
## Calculate the DM-Prune Lambda Path,
## given a user-specified prior
SHGR.trn$calc_DMPrune_LambdaPath(priorCADJ)
% Building DM-Prune Lambda Path:
% -----------------------------------------------------------------
% ++ gamma values: G = 1, N = 1, L = 1, S = 1
% No bootstrap edge significances found, skipping.
%    Call $set_BootSig to incorporate.
% Checking & storing prior ... done
```



```
% -----------------------------------------------------------------
% Computing mu scores:
% ++ mu_G (global) ... done
% ++ mu_N (local) ... done
% ++ mu_L (length) ... done
% ++ aggregate mu ... done
% ++ ranking mu ... done
% -----------------------------------------------------------------
% Computing Lambda Path ... done
```

The method `calc_DMPrune_LambdaPath` first stores the given prior, which can be accessed anytime via

```
## The prior values corresponding to the first few CADJ edges
head(c(SHGR.trn$DMP_prior))
```

```
% [1] 3.88246376 0.24885308 9.79543750 0.04908458 0.24244421 0.34280694
```

A scatter plot comparing each prior value to its corresponding CADJ count is automatically produced, allowing an analyst to assess the relationship between the prior and observed CADJ values. Intuitively, edges with low CADJ weight but high prior values are likely causing the most negative impact to DM-likelihood model.

Additionally, the individual and aggregate μ-scores, and the overall μ-rank for each edge, are computed and stored. These can be accessed via:

```
## For the first few CADJ edges ...

## Global mu-score
head(c(SHGR.trn$DMP_mu_G))
% [1] 0.30120482 0.02409639 0.33734940 0.01204819 0.01204819 0.01204819

## Local mu-score
head(c(SHGR.trn$DMP_mu_N))
% [1] 1.00000000 0.14285714 1.00000000 0.09090909 0.02777778 0.04000000

## Length mu-score
head(c(SHGR.trn$DMP_mu_L))
% [1] 1.000000000 0.965517241 1.000000000 1.000000000 0.001231527 0.001231527

## Composite mu-score
head(c(SHGR.trn$DMP_mu))
% [1] 0.670327915 0.149235209 0.696134749 0.103080474 0.007441968 0.008403792

## mu-rank, 1 = lowest composite mu-score
head(c(SHGR.trn$DMP_mu_rank))
% [1] 634 272 641 235  79  92
```

The (sorted) set of unique μ-ranks define the possible DM-Prune steps t are also stored

```
## DM-Prune steps
head(c(SHGR.trn$DMP_pruneStep))
% [1] 0 1 2 3 4 5
```

and the resulting Λ values (the normalized DM-likelihood) computed after sparsifying all edges from the model whose μ-rank is $<$ each step t can be accessed via

```
head(c(SHGR.trn$DMP_Lambda))
% [1] -2.961312 -2.961234 -2.961299 -2.961165 -2.961223 -2.959457
```

While the $\Lambda(t)$ values are available for extraction they are most useful when plotted via the `vis_DMPrune_LambdaPath` function, which takes a `TRN` object (whose Λ-path has been previously set) as input:

```
## Visualize the SHGR Lambda-path
TopoRNet::vis_DMPrune_LambdaPath(TRN = SHGR.trn)
% !! Lambda Path Visualization !!
% ----------------------------------------------------------
% max(Lambda) = -2.2 found at step = 467
% Lambda path intersects its starting value at step = 640
% Change points identified at steps:
% 268         577         640         673
% ----------------------------------------------------------
```

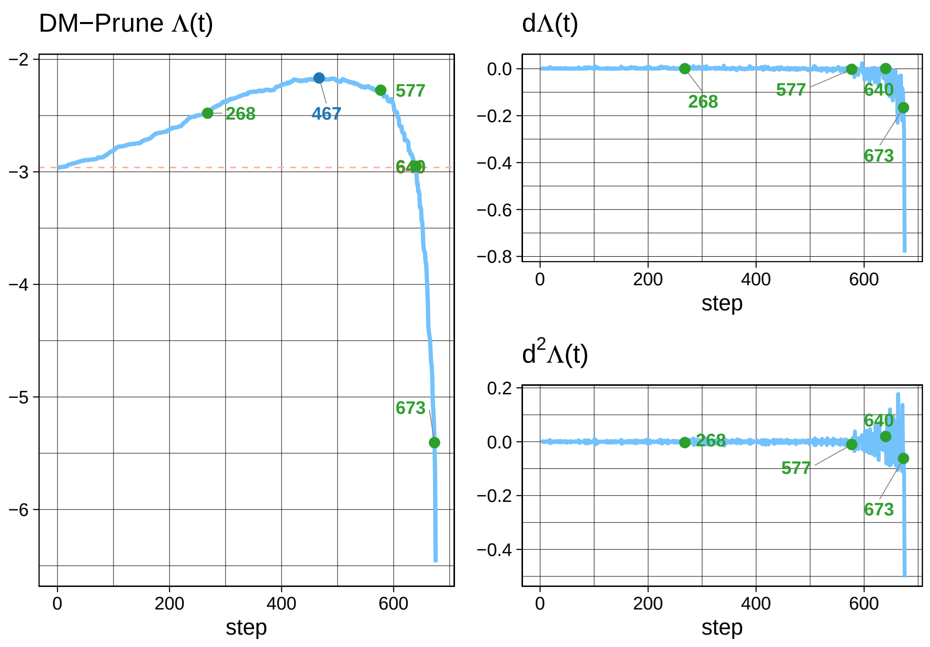


Figure 10: SHGR Λ Path

`vis_DMPrune_LambdaPath` also produces plots of the first and second-order finite differences of the Λ-path to further highlight points of (rapid) change in the path. Additionally, several time steps of interest (discussed below) are marked on the path as blue and green points.

The idea behind DM-Pruning is to successively remove CADJ edges until the DM-likelihood begins to noticeably deteriorate. The SHGR Λ-path displays a visible increase in likelihood as edges are removed, until step $t = 467$ (marked with a blue point). In statistics, parameterizations are often governed by the arg max of a model likelihood and the DM-Prune procedure recommends this step as a focal point for pruning decisions as well. Removing low-quality edges from the graph *should* improve the DM model fit (assuming the model is correctly specified), so the likelihood peak visible above is not entirely unsurprising. However, as work is ongoing to determine the fidelity and responsiveness of the DM-likelihood to salient cluster structure, alternative suggestions for an optimal pruning step are also marked in green. These alternative suggestions arise from variance changepoint analysis of the Λ-path and are intended to highlight pruning steps which precede larger swings in the path, signaling significant model changes. For the SHGR SOM, a conservative approach would stop pruning around step $t = 268$, while a more aggressive approach would continue pruning until step $t = 577$. For now, we suggest human assessment of several pruned CADJ graphs in this range. Below, we DM-Prune the CADJ graph at each of the suggested steps and compare the resulting CONNvis:

```
## DM-Prune at t=268
SHGR.trn$DMPrune_CADJ_step(268)
% Set 571 CADJ edges inactive.
## Propagate CADJ pruning to the CONN graph
SHGR.trn$prune_CONN_CADJ()
## View the resulting CONNvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()
SHGR.trn$restore_CONN_edges()

## DM-Prune at t=467
SHGR.trn$DMPrune_CADJ_step(467)
% Set 1250 CADJ edges inactive.
## Propagate CADJ pruning to the CONN graph
SHGR.trn$prune_CONN_CADJ()
## View the resulting CONNvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()
SHGR.trn$restore_CONN_edges()

## DM-Prune at t=577
SHGR.trn$DMPrune_CADJ_step(577)
% Set 1557 CADJ edges inactive.
## Propagate CADJ pruning to the CONN graph
SHGR.trn$prune_CONN_CADJ()
## View the resulting CONNvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CONNvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()
SHGR.trn$restore_CONN_edges()
```

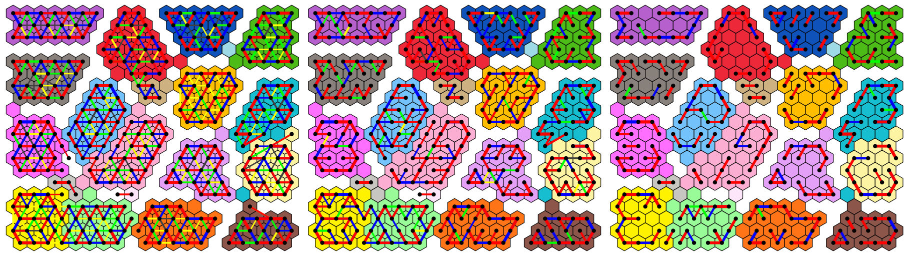

(a) t=268 (b) t=467 (c) t=577

Figure 11: DM-Pruned SHGR CONNvis

The conservative pruning at $t = 268$ is very similar to the grid-pruned CONN discussed above, failing to properly separate clusters F & M, K & N and D & T, while the aggressive pruning at $t = 577$ has left the graph mostly disconnected into tiny neuron islands. The pruning at $t^* = 467 = \arg\max \Lambda(t)$ has produced a graph with most of these problems rectified, except for one boundary violation between clusters F & M.

Note that in each group of function calls above we have first pruned the CADJ graph via the method `DMPrune_CADJ_step`, which takes as input a step t and sets all CADJ edges whose μ-rank is $<$ as inactive. The set of CONN edges corresponding to the resulting set of inactive CADJ edges in each case is inactived via `prune_CONN_CADJ`, and the CONNvis of the pruned CONN is visualized. This was done in order to view CONNvis, which is the more common visualization but only operates on the symmetric CONN graph. The CADJvis of the above could have been viewed instead, if preferred:

```
## DM-Prune at t=268
SHGR.trn$DMPrune_CADJ_step(268)
% Set 571 CADJ edges inactive.
## View the resulting CADJvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CADJvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()

## DM-Prune at t=467
SHGR.trn$DMPrune_CADJ_step(467)
% Set 1250 CADJ edges inactive.
## View the resulting CADJvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CADJvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
```

```
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()

## DM-Prune at t=577
SHGR.trn$DMPrune_CADJ_step(577)
% Set 1557 CADJ edges inactive.
## View the resulting CADJvis
SOMDisco::vis_som_label(SOM = SHGR.som, text.cex = 0)
SOMDisco::vis_som_CADJvis(SOM = SHGR.som, TRN = SHGR.trn, add = T)
## Restore all pruned edges
SHGR.trn$restore_CADJ_edges()
```

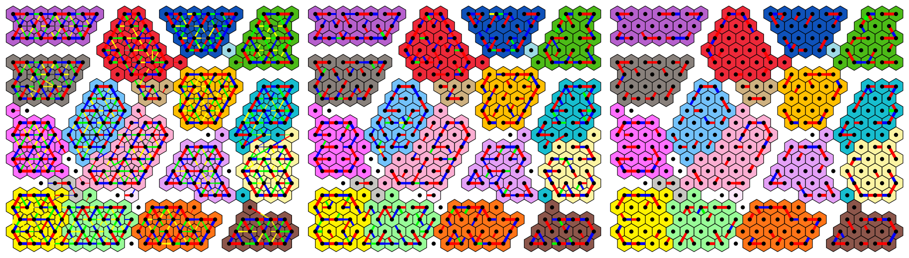

(a) t=268 (b) t=467 (c) t=577

Figure 12: DM-Pruned SHGR CADJvis

Note also that after each round of pruning all CADJ and CONN edges were restored. This was not necessary in this case as DM-Pruning is hierarchical, meaning that the pruned graph at step t_2 is a sub-graph of the graph pruned at t_1 if $t_2 > t_1$. In general, however, the effects of pruning a TRN (via any of the `prune_CADJ_*` or `prune_CONN_*` methods) should be reversed prior to re-pruning with a different set of criteria for proper comparison.

5 Topology Preservation Measures

Topology Preservation for TRN's with natural output space topology is complicated-sounding terminology for what is a fairly simple concept: the TRN should represent adjacencies in its input space by adjacencies in its output space *faithfully*. Any deviations from this are known as **topology violations**. Put another way, if two prototypes w_i and w_j are neighbors in $\mathcal{M}$ (e.g., $\exists$ an edge $CADJ_{ij} > 0$), their corresponding neurons ν_i and ν_j should **also be adjacenct** according to the topology of the output space $\mathcal{L}$. For SOM's $\mathcal{L}$ is obviously the lattice, whose topology is defined by the rigid rectangular or hexagonal grid structure. Measuring topology (violation) preservation is, then, equivalent to counting the number of neighbor relationships on $\mathcal{M}$ that are

(not) preserved by the TRN's representation on $\mathcal{L}$. To motivate this consider the following three cases to which we will return throughout this section:

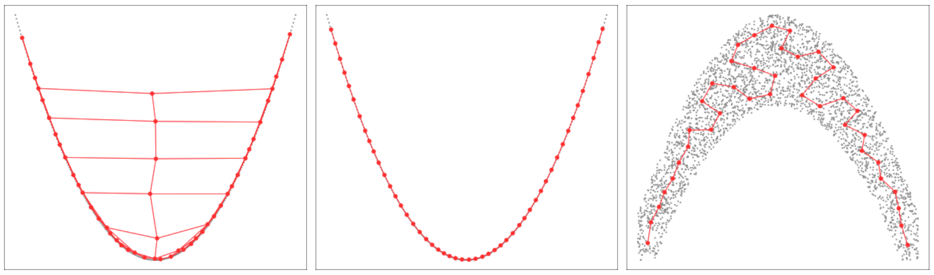

(a) A (b) B (c) C

Figure 13: Examples of 1-d TRN

Each panel depicts a sample of observations in 2-d input space (gray points) and the prototypes of a SOM trained on the sample (red points). The SOM lattice adjacencies (output space topology) are shown as red edges. The different effective dimensionalities of both input and output space in the above cases result in various types of topology violation:

- **Backward Topology Violations** (left panel, case "A") At left the sample data lie along a 1-d manifold $\subset \mathbb{R}^2$ but were learned by a SOM equipped with a 2-d rectangular output lattice. The dimension mismatch between these two spaces ($dim(\mathcal{L}) > dim(\mathcal{M})$) has caused the lattice to **fold** onto $\mathcal{M}$ such that neighboring lattice neurons are connected to prototypes which are *not* neighboring in $\mathcal{M}$ (as gauged by eye for now, this will be formalized below). This is known as a **backward** topology violation.

- **No Topology Violations** (middle panel, case "B") At middle the same sample data as in case A were learned by an SOM with a 1-d lattice (or chain of neurons). Since $dim(\mathcal{L}) = dim(\mathcal{M})$ we have no topology violations; the lattice adjacencies correspond perfectly to the adjacencies of the prototypes in $\mathcal{M}$.

- **Forward Topology Violations** (right panel, case "C") At right fully 2-dimensional sample data were learned by a SOM with a 1-d output space. This dimension mismatch ($dim(\mathcal{L}) < dim(\mathcal{M})$) has caused $\mathcal{M}$ to fold onto $\mathcal{L}$ (i.e., there are neighboring prototypes in $\mathcal{M}$ that are not neighbors on $\mathcal{L}$), which is typically the case with SOM learning.

The measures introduced in this section differ only in *how* and with *what granularity* they are sensitive to the various types of topology violations described above (i.e., by weighting or normalizing the counting of violations in various ways). We study a map's overall topology preservation and instances of topology violations in order to:

- monitor the progress of manifold learning (i.e., SOM training should persist until there is suitable representation of $\mathcal{M}$'s topology on $\mathcal{L}$)
- asses poor learning parameterizations (e.g., severely mis-specified learning rates governing SOM training)
- ensure *correct* (as possible) topological inference from the TRN (i.e., correctly identifying true cluster structure in the data, and avoiding the identification of spurious structure that might be indicated by violations)
- diagnose severe dimensionality conflicts between the TRN's input and output space topologies.

The remainder of this reviews several leading measures of topology preservation, which requires us to first clarify some terminology. Let:

- $\mathcal{M}$ be the data manifold as a (possible) subset of $\mathbb{R}^d$ (the input space)
- $\mathcal{L}$ be the output space ($\mathbb{R}^2$ with an embedded lattice structure for SOMs)
- $d_a^b(i,j)$ be the distance defined by a measured on space b between vertices i and j of the TRN.
  $b \in \{\mathcal{M}, \mathcal{L}\}$ defines the space in question. $a \in \{E, G\}$ where E is Euclidean distance and G is the geodesic distance on the corresponding graph inherent to space b (e.g., CADJ on $\mathcal{M}$ and the lattice for $\mathcal{L}$).
- $\eta(i,k,a,b)$ return the k-th nearest neighbor of vertex i in space b according to distance a.

5.1 Topographic Product

The Topographic Product (TP, [11]) provides a *global* measure of topology preservation through multiplicative accumulation of both backward and forward toplogy violations. This occurs through two intermediary calculations representing ratios of distances between equivalent quantities in our different spaces:

$$Q_1(i,k) = \frac{d_E^{\mathcal{M}}(w_i, w_{\eta(i,k,E,\mathcal{L})})}{d_E^{\mathcal{M}}(w_i, w_{\eta(i,k,E,\mathcal{M})})}$$

measures the ratio of Euclidean distances between neighboring prototypes in input space, where the different concepts of "neighbor" (according to both the output and input space topologies) define the ratio. The quantity Q_2 is similar, except that the Euclidean distances are calculated in output space (instead of input space):

$$Q_2(i,k) = \frac{d_E^{\mathcal{L}}(w_i, w_{\eta(i,k,E,\mathcal{L})})}{d_E^{\mathcal{L}}(w_i, w_{\eta(i,k,E,\mathcal{M})})}.$$

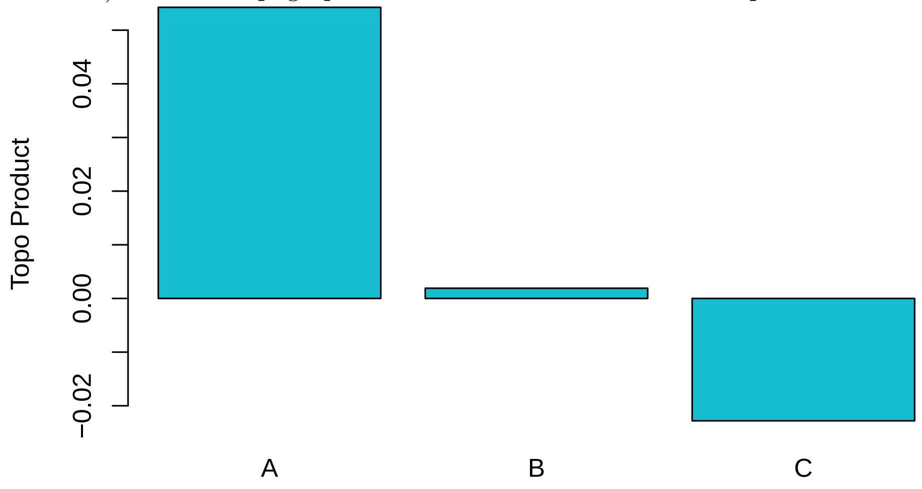
Q_1 measures *backward* violations while Q_2 measures *forward* violations; if there are none then $Q_1 = Q_2 = 1$. The ratios are combined into the quantity

$$Q_3(i,k) = \left(\prod_{l=1}^{k} Q_1(i,l) \times Q_2(i,l) \right)^{\frac{1}{2k}}$$

which measures the overall type of folding from prototype / neuron i occuring at folding length k. Aggregating this quantity over all possible folding lengths allowed for given the input / output topologies, and all neurons present in the lattice, gives the Topographic Product (TP):

$$TP = \frac{1}{n_W(n_W - 1)} \sum_{i=1}^{n_W} \sum_{k=1}^{n_W - 1} \log(Q_3(i,k))$$

The **sign** of the TP indicates the overall type of topology violations of the TRN: $TP > 0 \Rightarrow$ backward violations (case A from above), $TP \approx 0 \Rightarrow$ no violations (case B above), and $TP < 0 \Rightarrow$ forward violations (case C above). The Topographic Products of our three example cases are:



Case A has TP = 0.0542559, case B has TP = 0.0019023 while case C has TP = -0.0227994. Earlier we identified each as exhibiting backward, no, and forward topology violations, respectively. The Topographic Product confirms this assessment, based on sign.

5.2 Differential Topographic Function

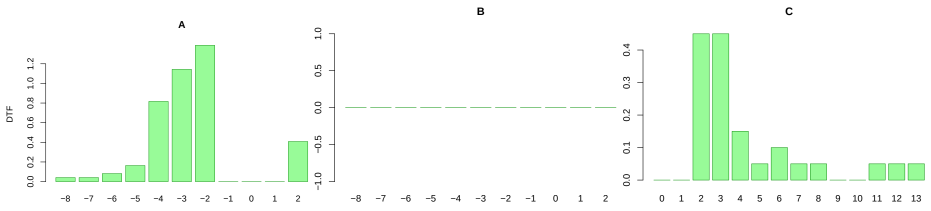
The geodesic distance of connections in the output (input) topology when represented on the input (output) topology is known as the **folding length** of a backward (forward) topology violation. By convention, backward violations have a negative folding length, while forward violations have a positive folding length. The Differential Topographic Function (DTF, [12]) counts the number of violating connections at each folding length $\in [-k, \dots, k]$. The intermediate quantity $f_i(k)$ is defined different based on the sign of its folding length argument

$$f_i(k) = \begin{cases} \sum_{j=1}^{n_W} \mathbb{1}[d_G^{\mathcal{L}}(i,j) = k \text{ and } d_G^{\mathcal{M}}(i,j) = 1] & k > 0 \\ \sum_{j=1}^{n_W} \mathbb{1}[d_G^{\mathcal{L}}(i,j) = 1 \text{ and } d_G^{\mathcal{M}}(i,j) = |k|] & k < 0 \end{cases}$$

yielding the DTF:

$$DTF(k) = \frac{1}{n_W} \sum_{i=1}^{n_W} f_i(k)$$

As folding lengths = 1 are not technically topology violations, $DTF(k) := 0$ for $k \in [-1, 0, 1]$. The DTF of each of the above cases plotted over its apparent folding lengths of each of the cases above is:



The DTF again confirms the conclusion that case A exhibits (mostly) backward violations, case B exhibits no violations, and case C exhibits forward violations.

5.3 Weighted Differential Topographic Function

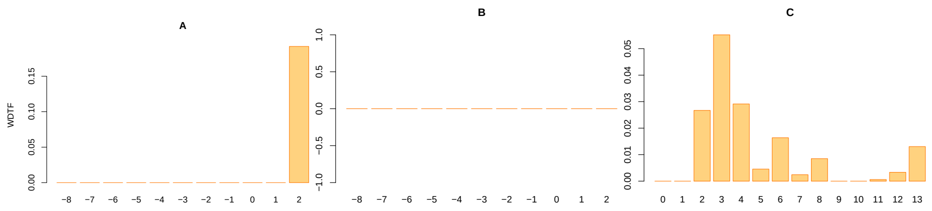
If the input topology adjacencies are weighted (as is CADJ) the count of violating connections (either backward of forward) identified by the DTF can be multiplied by these weights, producing the Weighted Differential Topographic Function (WDTF, [12]). Using CADJ as the input topology, the f function above is modified as

$$f_i'(k) = \sum_{j=1}^{n_W} CADJ_{ij} \times \mathbb{1}[d_G^{\mathcal{L}}(i,j) = k \text{ and } d_G^{\mathcal{M}}(i,j) = 1].$$

Notice that the backward folding lengths are not captured, as CADJ only represents the forward mapping. The WDTF at each positive folding length is then

$$WDTF(k) = \frac{1}{\sum CADJ} \sum_{i=1}^{n_W} f_i'(k)$$

where the normalizing constant is the sum of all edge weights in CADJ. The WDTF of our three examples confirms the conclusions of the DTF (where possible), particularly in case C. From the WDTF we also learn that some relatively strong connections are forward violating at large folding length:

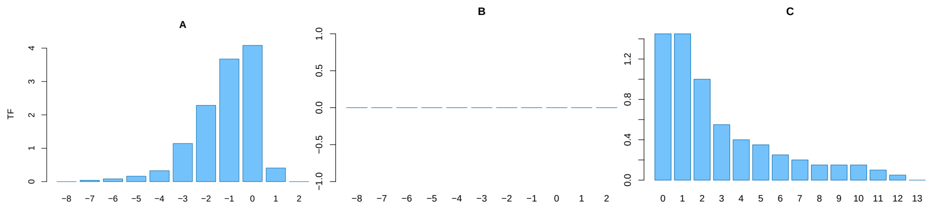


5.4 Topographic Function

The Topographic Function (TF, [13]) is an integral representation of the DTF reporting, for each folding length k, the cumulative violations of length larger than k (whether they be backward or forward):

$$TF(k) = \begin{cases} \sum_{k^{`}=k+1}^{n_W} DTF(k^{`}) & k > 0 \\ \sum_{k^{`}=-n_W}^{k-1} DTF(k^{`}) & k < 0 \end{cases}$$

The TF conveys less detail than the DTF but is included in `TopoRNet` for completeness. For our three cases, the TF reports:



Evidence of the primarily backward violations of case A and forward violations of B is (perhaps) easier to spot in the TF, but analysts generally would be more interested in knowing the *exact* length of a violating connection, not a range.

5.5 Calculating TPMs with TopoRNet

As the calculations of topology preservation measures introduced in this section are relatively lightweight `TopoRNet` computes them all during a call to the method `calc_TopoMeasures`, which takes as input a matrix of SOM prototype vectors in the TRN's input space (each $\in \mathbb{R}^d$) and a matrix of the neuron coordinates in output space (each usually in $\in \mathbb{R}^2$ for SOMs). These matrices should be ordered such that row i and row j give the prototype (neuron) coordinates of the vertices connected by the edge $CADJ_{ij}$, as set during `initialize_TRN`. The input / output topologies (e.g., CADJ and the lattice adjacency for SOMs) used for the TF, DTF and WDTF calculation were stored during `initialize_TRN` and re-used for these calculations; the inputs to `calc_TopoMeasures` are needed for TP calculation. Below we compute all TPMs for our SHGR example using the information stored in the SHGR SOM object previously loaded:

```
## Compute TPMs
## Inputs: SOM prototypes, (x,y) coords of neurons on lattice
SHGR.trn$calc_TopoMeasures(SHGR.som$W, SHGR.som$nu_xy)
% Computing Toplogy Preservation Measures:
% ++ Topographic Product ... done
% ++ CADJ Topographic Functions ... done
% ++ CONN Topographic Functions ... done
```

The TP result is stored as a scalar field which can be retrieved via

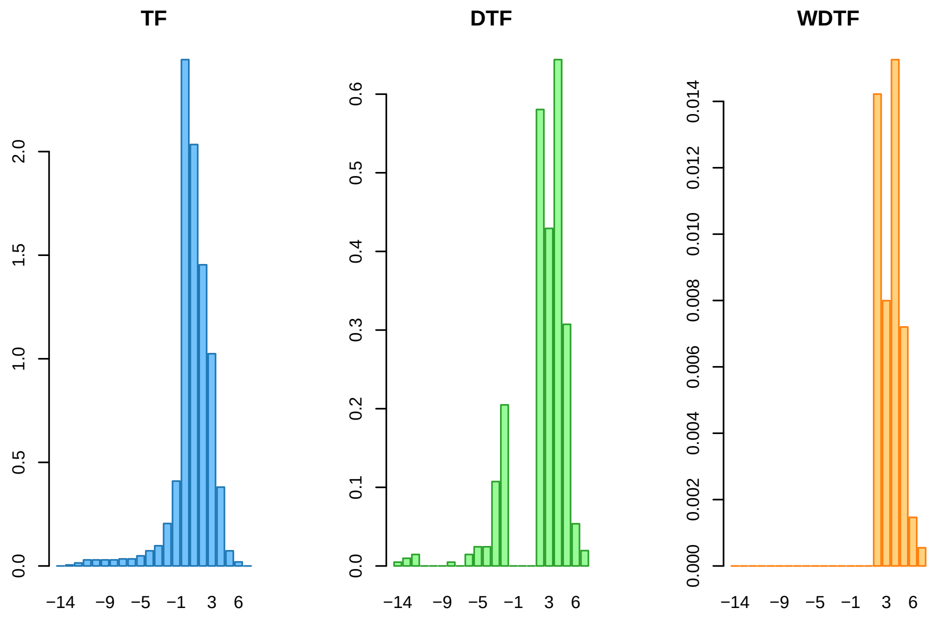
```
## Topographic Product result
SHGR.trn$TopoProd
% [1] 0.008247703
```

while the various Topographic Functions are stored as a data frame whose rows report each measure computed at the various folding lengths apparent in the mapping. Here, we extract the TFs computed using the CONN graph:

```
## The CONN Topographic Functions
SHGR.trn$CONN_TopoFxns
%      k          TF         DTF         WDTF
% 1  -14 0.000000000 0.004878049 0.0000000000
% 2  -13 0.004878049 0.009756098 0.0000000000
% 3  -12 0.014634146 0.014634146 0.0000000000
% 4  -11 0.029268293 0.000000000 0.0000000000
% 5  -10 0.029268293 0.000000000 0.0000000000
% 6   -9 0.029268293 0.000000000 0.0000000000
% 7   -8 0.029268293 0.004878049 0.0000000000
% 8   -7 0.034146341 0.000000000 0.0000000000
% 9   -6 0.034146341 0.014634146 0.0000000000
% 10  -5 0.048780488 0.024390244 0.0000000000
% 11  -4 0.073170732 0.024390244 0.0000000000
% 12  -3 0.097560976 0.107317073 0.0000000000
% 13  -2 0.204878049 0.204878049 0.0000000000
% 14  -1 0.409756098 0.000000000 0.0000000000
% 15   0 2.443902439 0.000000000 0.0000000000
% 16   1 2.034146341 0.000000000 0.0000000000
% 17   2 1.453658537 0.580487805 0.0142211914
% 18   3 1.024390244 0.429268293 0.0079956055
% 19   4 0.380487805 0.643902439 0.0152587891
% 20   5 0.073170732 0.307317073 0.0072021484
% 21   6 0.019512195 0.053658537 0.0014648438
% 22   7 0.000000000 0.019512195 0.0005493164
```

`TopoRNet` also provides a visualization for the TFs computed and stored in a `TRN` object:

```
TopoRNet::vis_CONN_TopoFunctions(TRN = SHGR.trn)
```



References

[1] T. Martinetz and K. Schulten, "Topology representing networks," *Neural Networks*, vol. 7, no. 3, pp. 507–522, 1994.

[2] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, pp. 793–800, 1934.

[3] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu, *Spatial tessellations: Concepts and applications of Voronoi diagrams*, 2nd ed. John Wiley; Sons, Inc., 2000.

[4] K. R. Gabriel and R. R. Sokal, "A new statistical approach to geographic variation analysis," *Systematic Zoology*, vol. 18, no. 3, pp. 259–278, 1969.

[5] G. T. Toussaint, "The relative neighbourhood graph of a finite planar set," *Pattern recognition*, vol. 12, no. 4, pp. 261–268, 1980.

[6] K. Taşdemir and E. Merényi, "Exploiting data topology in visualization and clustering of self-organizing maps," *IEEE Transactions on Neural Networks*, vol. 20, no. 4, pp. 549–562, 2009.

[7] T. Kohonen, *Self-organizing maps.* Springer, 2000.

[8] E. Merényi, K. Tasdemir, and L. Zhang, "Learning highly structured manifolds: Harnessing the power of SOMs," in *Similarity-based clustering*, vol. 5400,

M. Biehl, B. Hammer, M. Verleysen, and T. Villmann, Eds. Berlin Heidelberg: Springer Verlag, 2009, pp. 138–168.

[9] J. Taylor and E. Merényi, "A probabilistic method for pruning cadj graphs with applications to som clustering," in *Advances in self-organizing maps, learning vector quantization, clustering and data visualization*, 2020, pp. 44–54.

[10] J. Taylor and E. Merényi, "DM-pruning cadj graphs for som clustering," *Neural Computing and Applications*.

[11] H.-U. Bauer, K. Pawelzik, and T. Geisel, "A topographic product for the optimization of self-organizing feature maps," in *Advances in neural information processing systems 4*, J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds. Morgan-Kaufmann, 1992, pp. 1141–1147.

[12] L. Zhang and E. Merényi, "Weighted differential topographic function: Arefinement of the topographic function," in *In proc. 14th european symposium on artificial neural networks (esann'2006*, 2006, pp. 13–18.

[13] T. Villmann, R. Der, M. Herrmann, and T. M. Martinetz, "Topology preservation in self-organizing feature maps: Exact definition and measurement," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 256–266, 1997.