GRAPH NUERAL NETWORKS AND APPLICATIONS TO REEB GRAPHS

BRIAN BOLLEN

Abstract. The Reeb Graph is a topological description of a scalar fields. Often, Reeb Graphs are used to summarize "cirtical information" from a data set. Comparing two Reeb Graphs to each other, however, remains to be an open problem in the Topological Data Analysis (TDA) community. Current distance metrics such as the Interleaving Distance suffer from an exponential computation time. With the increase in computing power in recent years, it is hopeful that we can leverage machine learning techniques to construct a metric on Reeb Graphs that is easier to compute. In this paper, we review two approaches for classifying graphs via Convolutional Graph Neural Networks (ConvGNN). Our goal is to begin a discussion onto whether these approaches are useful to classify Reeb Graphs. We will show that the Spatial ConvGNN will most likely provide a more useful architecture for Reeb Graphs due to their flexibility over the Spectral architecture.

.

1. Introduction

Determining whether two graphs are similar to one another continues to be a challenging problem in the mathematical community. It is known that determining if two graphs are isomorphism is an NP-hard problem. A lot of work has been done in order to develop metrics that indicate how close graphs are to one another. In addition to this, graphs may arise from many different scenarios, where the graph is a representation of the original data set. In this case, the goal is to determine how close the graphs are to one another in a way that provides us information about the underlying data set. Because of this, the choice of metric used on these graphs is important.

The Reeb Graph is a multigraph which encompasses important topological features of a scalar field. It shows us how many connected components exist at any scalar value a. In this way, it shows us the critical points, where connected components split or merge.

A lot of work has been done in the area of determining distance between Reeb Graphs, as well as the distance between other topological descriptors such as Persistence Diagrams. However, distances which have been shown to be mathematically fruitful, such as the Interleaving Distance, have proven to still be extremely difficult to compute.

Consider multiple simulations of a weather phenomenon. Suppose each simulation can be represented as a scalar field. That is, each simulation is a pair $(\mathbb{X}, f)$

where $\mathbb{X}$ is a topological space and $f : \mathbb{X} \to \mathbb{R}$ is a continuous scalar function. Between simulations, we might have slightly different results. Our goal would be to determine which would be the "best" simulation, or possibly an "average" of all the simulations. To do this, we need to develop some sort of notion of distance on the ensemble of scalar fields. We could first convert each scalar field into is Reeb Graph which will be an efficient and informative descriptor of the important events of the scalar field. Then, we would use a metric between the Reeb Graphs which will hopefully provide useful information about the scalar fields that they arise from.

With the rise of Machine Learning, it is of interest to see if we can leverage contemporary machine learning techniques to classify these Reeb Graphs. If we can classify the Reeb Graphs appropriately, we might be able to construct and embedding for the Reeb Graphs where we can then define a distance between them. One of our ideas is to use Convolutional Graph Neural Networks. A convolutional nueral network is a neural network where one or more layers involve the mathematical operation of convolution. CNN's are often used in image classification where each neuron of the network is given a **receptive field**, which is the set of neurons from which it receives input. This is analogous to how real biological neurons act in the visual cortex; the neurons fire depending on input from specific portions of field of view of the organism.

The paper is split into two sections: Spectral Convolutional GNNs and Spacial Convolutional GNNs. Both are cornerstone methods for classification of graphs using neural networks. While the spectral methods have a strong mathematical foundation in graph signal processing, spatial methods have been becoming more popular as it has been shown that spatial methods are more general, flexible, and efficient [4].

This work is a begining literature review of Neural Networks applied to Graphs. **The goal of this analysis is to provide multiple different methods for how graphs are classified using Neural Networks**. We will discuss why Spatial GNNs are now favored over Spectral GNNs due to multiple limitations. Hopefully, with this insight, we can begin to move towards leveraging machine learning techniques to define distances on Reeb Graphs rather than use the currently difficult to compute metrics such as interleaving distance [3]. Note that this work only discusses using GNN's for classification of the Reeb Graphs, not necessarily constructing a well-defined metric (or psuedo-metric) on the graphs.

2. Preliminary Definitions

Definition 2.1. A **graph** G is a pair (V, E) of vertices $V = \{v_1, \ldots, v_n\}$ (or nodes) and edges $E \subseteq V \times V$ between the vertices. Note that there need not exist an edge between each pair of vertices. A **multigraph** is a graph G such that the set of edges is a multiset. That is, there can be multiple edges between the same two vertices. The **adjacency matrix A** is a matrix where $\mathbf{A}_{ij} = n$, where n is the number of edges between vertices v_i and v_j. Note that if the graph G is not a multigraph, $\mathbf{A}_{ij}$ is either 1 or 0.

Definition 2.2. Let v_1, v_2 be two vertices in a graph. Then we say that $d(v_1, v_2)$ is equal to the length of the shortest path between v_1 and v_2.

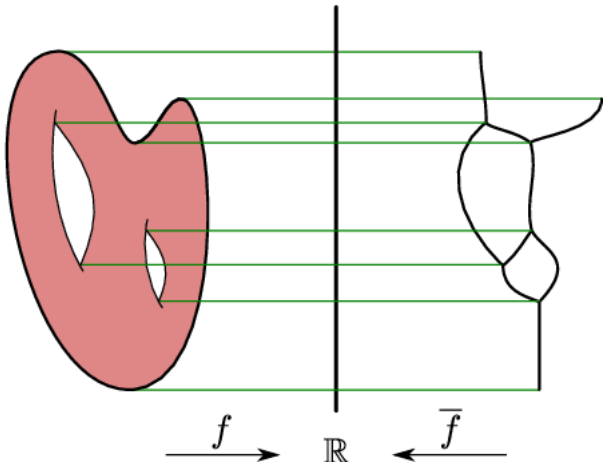

FIGURE 1. A two manifold with a scalar function mapping its height to the real line accompanied with its Reeb Graph. We can see that this forms a well-defined multigraph. The figure can be found in [3]

Definition 2.3. Let $\mathbb{X}$ be a topological space and $f : \mathbb{X} \to \mathbb{R}$ be a continuous scalar function. Then, the pair $(\mathbb{X}, f)$ is a **scalar field**. We say that the point pre-images $f^{-1}(a)$ are **level sets** or a-**fibers**.

Definition 2.4. Let $(\mathbb{X}, f)$ be a scalar field. We define an equivalence relation $\sim_f$ as follows: $x \sim_f y$ if $f(x) = f(y) = a$ and if x and y belong to the same path-connected component of the pre-image $f^{-1}(a)$. We say that the quotient space $\mathbb{X}_f := \mathbb{X}/\sim_f$ is the **Reeb Graph** of the scalar field $(\mathbb{X}, f)$.

Figure 1 provides an example of what a Reeb Graph is. While generally the Reeb Graph is thought of as a quotient space, we can combinatorially think about it as being a multigraph. With the image on the right, we would consider all points where edges connect to be vertices.

3. Spectral Convolutional GNNs

Spectral Convolutional GNNs use techniques from graph signal processing and have a strong mathematical foundation. It defines a convolution using a **filter** which is used to remove noise from a graph signal.

Definition 3.1. Let G be a graph, $\mathbf{A}$ be its adjacency matrix, and $\mathbf{D}$ be a diagonal matrix of node degrees. That is, $D_{ii} = \sum_j \mathbf{A}_{ij}$. Then, we define the **graph Laplacian** as $\mathcal{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$. In some works, this is known as the **normalized graph Laplacian**, where the regular graph laplacian is defined simply as $\mathcal{L} = \mathbf{D} - \mathbf{A}$. The Laplacian is seen to be a matrix representation of an undirected graph.

The normalized graph Laplacian is a symmetric, positive, semi-definite matrix. Because of this, it can be diagonalized and rewritten as $\mathcal{L} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, where $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_n]$ and each $\mathbf{u}_i$ is an eigenvector, and $\mathbf{\Lambda}$ is the diagonal matrix of

corresponding eigenvalues.

Definition 3.2. The **Graph Fourier Transform** of the vector $\mathbf{x}$ is defined as $\mathcal{F}(\mathbf{x}) = \mathbf{U}^T\mathbf{x}$ and the **Inverse Graph Fourier Transform** is defined as $\mathcal{F}^{-1}(\mathbf{x}) = \mathbf{U}(\mathbf{x})$

The Convolution Theorem states that the Fourier transform of the convolution of two operators is equivalent to the product of the Fourier transforms of the operators individually. That is, $\mathcal{F}(f \star g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$. Combining this fact with the definition above, we define the graph convolution with a filter $\mathbf{g} \in \mathbb{R}^n$ as

$$\mathbf{x} \star_G \mathbf{g} = \mathcal{F}^{-1}(\mathcal{F}(\mathbf{x}) \cdot \mathcal{F}(\mathbf{g})) = \mathbf{U}(\mathbf{U}^T\mathbf{x} \cdot \mathbf{U}^T\mathbf{g}),$$

where $\cdot$ denotes the pointwise product. Let $g_\theta = diag(\mathbf{U}^T\mathbf{g})$, where $diag(textbfc)$ creates a diagonal matrix $\mathbf{A}$ where $\mathbf{A}_{ii} = c_i$. Then, we have

$$\mathbf{x} \star_G \mathbf{g}_\theta = \mathbf{U}\mathbf{g}_\theta\mathbf{U}^T\mathbf{x}$$

All spectral based approaches use this definition for a convolution. The only difference is the choice of the filter $\mathbf{g}_\theta$ [4].

In [1] the authors assume that $g_\theta = \mathbf{\Theta}_{i,j}^{(k)}$ is a set of learnable parameters. The graph convolutional layer in this setup is defined as

$$\mathbf{H}_{:,j}^{(k)} = \sigma\left(\sum_{i=1}^{f_{k-1}} \mathbf{U}\mathbf{\Theta}_{i,j}^{(k)}\mathbf{U}^T\mathbf{H}_{:,j}^{(k-1)}\right) \quad \text{for} \quad j = 1, 2, \ldots, f_k,$$

where k is the layer index, $\mathbf{H}^{(k-1)} \in \mathbb{R}^{n \times f_{k-1}}$ is the input graph signal, $\mathbf{H}^{(0)} = \mathbf{X}$, f_{k-1} is the number of input channels and f_k is the number of output channels.

4. Spatial Convolutional GNNs

A Spatial Convoluation GNN uses properties of the neighborhood of a node to update the state for the node in question. This is much more direclty related to CNNs used on images. Suppose we have an image of arbitrary size. With a convolutional neural network, we take a single pixel and update its state based on pixels around it. The pixels around it are known as that pixels **receptive field**. Suppose the receptive field for each pixel is 3×3. When calculating the weighted average, it depends specifically on the placement of the pixel in relation to the center. So, in essence, we end up taking a single pixel and define an ordering to the pixels around it. Perhaps starting in the upper left, we label that as 1 and then continue in a clockwise fashion until we label each of the 8 surrounding pixels.

An image can be thought of as a very specific form of graph where each pixel is a node with connections to each pixel adjacent to it. So, we can revamp our definition of receptive field a bit to say that we update the nodes state based on its *neighborhood*. Since an image is a very predictable form of graph, defining what this neighborhood is is quite easy: Let x be a pixel in an image. We say that $N_1(x)$ is the set of pixels that are adjacent to x. In graph theoretic terms, we say that x is a node and $N_1(x)$ is the set of nodes V such that $d(x, v) = 1$ for each $v \in V$.

This generalizes for $N_n(x)$ with distance n.

Now, the labeling procedure we constructed for images becomes quite complicated of a question. When considering constructing the receptive field for a single node, do we always try to have the same size receptive field for each node? If so, how do we construct this? Of course, in an arbitrary graph, each node will have varying sizes of $N_1(x)$.

Definition 4.1. Let $G = (V, E)$ be a graph. A graph **labeling** l is a function $l : V \to S$ where S is an ordered set (such as the integers).

Definition 4.2. A **ranking** is a function $\mathbf{r} : V \to \{1, \ldots, |V|\}$ Every labeling induces a ranking $\mathbf{r}$ with $\mathbf{r}(u) < \mathbf{r}(v)$ if and only if $l(u) > l(v)$.

In [2], the authors describe their program PATCHY-SAN which is used to classify Graphs using a spatial convolutional GNN architecture. The SAN in PATCHY-SAN stands for "Select, Assemble, Normalize". Here, we describe the listed procedures.

4.1. **Node Sequence Selection.** In an image, not only is the receptive field intrinsically ordered, but the image as a whole is ordered as well. With graphs, we need to have some place to start. All of this comes down to a labeling procedure. Different labeling procedures are used such as node degree or attribute value.

The node sequence selection algorithm described below is responsible for determining the sequence of nodes for which receptive fields are created. We suppose that we have w input channels, each of which must have a receptive field created for it.

Algorithm 1 SELNODESEQ: Select Node Sequence

input: graph labeling procedure l, graph $G = (V, E)$, stride s, width w, receptive field size k
$V_{\texttt{sort}} =$ top w elements of V according to l
$i = 1, j = 1$
while $j < w$ **do**
  if $i < w$ **then**
    $f =$ RECEPTIVEFIELD($V_{\texttt{sort}}[i]$)
  else
    $f =$ ZERORECEPTIVEFIELD()
  end if
  apply f to each input channel
  $i = i + s$,$j = j + 1$
end while

The stride s determines the distance between consecutive nodes with respect to the labeling (not to be confused with the shortest path distance between nodes). If i surpasses the top w elements of nodes and there are still receptive fields needed to be created, we add an all-zero receptive field instead.

4.2. **Receptive Field Creation.** The receptive fields are created using another algorithm described below:

Algorithm 2 ReceptiveField: Create Receptive Field

input: vertex v, graph labeling l, receptive field size k
$N = \text{NeighAssemb}(v, k)$
$G_{\texttt{norm}} = \text{NormalizeGraph}(N, v, l, k)$
return $G_{\texttt{norm}}$

In the algorithm above, we have two more algorithms: The NeighAssemb algorithm is used to construct the candidates for the receptive field for a given vertex v. After the neighborhood is assembled, NormalizeGraph is used to imopose an order on the nodes of the neighborhood graph so we can map from the unordered graph space to a vector space with linear order.

Algorithm 3 NeighAssemb: Neighborhood Assembly

input: vertex v, receptive field size k
output: set of neighborhood nodes N for v
$N = [v]$
$L = [v]$
while $|N| < k$ and $|L| > 0$ **do**
    $L = \cup_{v \in L} N_1(v)$
    $N = N \cup L$
end while
return the set of vertices N

4.3. **Neighborhood Assembly.** The neighborhood assembly algorithm first defines N and L as lists only containing the vertex v. We then add the 1-neighborhood of v to the list N. If the size of N is less than the indicated size of receptive field, then we add the 2-neighborhood of v to N and continue the process. In the process, the receptive fields for each input channel can vary in size.

4.4. **Graph Normalization.** While we can define a labeling for the entire graph using a labeling process, it is not as straightforward to construct rankings for a neighborhood. We must take into account that the nodes in a 1-neighborhood should be weighted differently than those in a 2 or 3-neighborhood. Given a subset of vertices U (from the neighborhood assembled), we compute the ranking based on the labeling procedure l (which need not be the same labeling procedure as used in the node sequence selection). However, we provide a stipulation that if the an element is in the 1-neighborhood of the vertex, then it must rank higher than those in the 2 or 3-neighborhood. Afterwords, depending on the size of U, we either remove nodes or add in dummy nodes. By the end, each of the normalized neighborhoods will be of the same size k.

5. Discussion

5.1. **Spectral vs Spatial.** The main issue with the spectral approach (especially when considering our scenario) is one of flexibility. This spectral approach is specifically designed to consider a $n \times m$ grid with values at each point. We can actually think of this as simply a discretized scalar field, perhaps. If we are only looking at

Algorithm 4 NormalizeGraph: Graph Normalization

input: subset of vertices U from original graph G, vertex v, graph labeling l, receptive field size k.
output: receptive field for v
compute ranking $\mathbf{r}$ of U using l, subject to $\forall u, w \in U : d(u, v) < d(w, v) \Rightarrow \mathbf{r}(u) < \mathbf{r}(v)$
if $|U| > k$ **then**
    $N =$ top k vertices in U according to $\mathbf{r}$
    compute ranking $\mathbf{r}$ of N using l, subject to
    $\forall u, w \in N : d(u, v) < d(w, v) \Rightarrow \mathbf{r}(u) < \mathbf{r}(w)$
else if $|U| < k$ **then**
    $N = U$ and $k - |U|$ dummy nodes
else
    $N = U$
end if
construct subgraph $G[N]$ for the vertices N
canonicalize $G[N]$, respecting the prior ranking $\mathbf{r}$
return $G[N]$

these scalar fields where the domains are exactly the same, but the values might be different, then this spectral approach has no immediate issue. However, if we are trying to compare two graphs with different Laplacian matrices, then the eigenbasis $\mathbf{U}$ is different as well.

The spectral approach seems like it could be useful if we were considering just the scalar field itself. As we discussed before, a common situation is having multiple simulations ran on the same data. In this case, our domains will be the same and therefore the graph we feed into the Neural Network is also the same. Thus, the spectral approach doesn't run into the issue of flexibility. It is worth looking into these approaches specifically on the scalar field rather than the Reeb Graph.

In contrast, the spatial architecture seems readily accessible to Reeb Graphs. The spatial approach can be used to classify graphs with different topological structures, which is exactly what we need. The tricky part for this approach is still a choice of labeling.

5.2. **Application to Reeb Graphs.** In [2], they bring up Graph Normalization Problem: "What is the best way to label this graph?". Mathematically, it is written as follows:

Let $\mathcal{G}$ be a collection of unlabeled graphs with k nodes, let l be a graph labeling procedure, let d_G be a distance measure between graphs of k nodes, and let d_A be a distance measure on $k \times k$ matrices. We seek to find $\hat{l}$ such that

$$\hat{l} = \arg\min_{l} \mathbb{E}_{\mathcal{G}}[|d_A(\mathbf{A}^l(G), \mathbf{A}^l(G') - d_G(G, G')|] \tag{5.1}$$

When trying to apply this spatial architecture to Reeb Graphs, we again run into the issue of distance between graphs: What is the best way to compare two Reeb Graphs? There has been some work in attempting to label a variant of Reeb

Graphs known as **Merge Trees** [5], but even in this case, there are many heuristics involved.

5.3. **Minimal Working Example of Receptive Field Creation.** To begin with work on constructing examples for Reeb Graphs, a minimal working example has been provided for creating Receptive fields on a graph. As for labeling, we have simply assumed that a higher value for the vertex indicates a higher rank. While this is a quite simple (and naive) approach for labeling, it makes some sense in terms of scalar fields. Given a Reeb Graph, we have an intrinsic ordering on the vertices (critical points) by associating the label with the function value. The only thing we need to worry about is breaking ties.

The example is written completely in Javascript. No external packages are required to run. The code was run on Node.js version 13.14.0, but any version of Node.js should suffice.

6. Conclusion and Future Directions

Spatial and Spectral Convolution Graph Neural Networks both have a place in the Machine Learning community. When it comes to classifying graphs with varying structure, however, the Spatial architecture clearly wins the race. When attempting to apply this to Reeb Graphs (or any other area), it is of upmost important that we choose a labeling algorithm wisely. Unfortunately, correct labeling of graphs continues to be a difficult problem.

As Patchy-san does, it might be possible to use multiple different labeling algorithms (if labeling is computationally easy enough) and see which one proves to be most useful.

In other works, the Interleaving Distance has proven to be useful inspiration for other metrics, even if it is itself difficult to compute. Perhaps we will discover that using the Interleaving Distance (theoretically) provides a canonical way to label Reeb Graphs that produces the most optimal label in 5.1.

References

[1] Joan Bruna et al. "Spectral networks and locally connected networks on graphs". In: *CoRR* abs/1312.6203 (2014). arXiv: 1312.6203. URL: https://arxiv.org/abs/1312.6203.

[2] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. "Learning Convolutional Neural Networks for Graphs". In: *CoRR* abs/1605.05273 (2016). arXiv: 1605.05273. URL: http://arxiv.org/abs/1605.05273.

[3] Vin de Silva, Elizabeth Munch, and Amit Patel. "Categorified Reeb Graphs". In: *CoRR* abs/1501.04147 (2015). arXiv: 1501.04147. URL: http://arxiv.org/abs/1501.04147.

[4] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596. URL: http://arxiv.org/abs/1901.00596.

[5] Lin Yan et al. "A Structural Average of Labeled Merge Trees for Uncertainty Visualization". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020), pp. 832–842. ISSN: 2160-9306. DOI: 10.1109/tvcg.2019.2934242. URL: http://dx.doi.org/10.1109/TVCG.2019.2934242.

Current address: Department of Mathematics, University of Arizona, Tucson, Arizona 85719
Email address: bbollen23@math.arizona.edu