Topological Interpretability of Neural Network Latent Spaces via Persistent Homology

Aum Rawal
github.com/aumrawal/repo

Abstract

Deep neural networks map data into opaque latent manifolds. To reverse-engineer a classifier's 256-dimensional hidden layer, we apply Topological Data Analysis (TDA). Crucially, we avoid external libraries, building persistent homology algorithms entirely from scratch using explicit simplicial complexes, boundary matrices, and matrix reduction. Consequently, this paper serves as a practical guide for beginners learning TDA. By tracking 0-dimensional components and 1-dimensional voids, we show how networks isolate classes, identifying a high-persistence loop bridging digits 9, 9, and 6. We propose extending these foundational methods to non-Euclidean datasets like the cosmic web.

1 Introduction

Neural networks compress high-dimensional raw pixel space—which is overly rigid and susceptible to the curse of dimensionality—into lower-dimensional latent spaces. While the performance of these networks is easily quantified, the underlying geometry of their "decision manifolds" is difficult to interpret. This project bridges that gap using computational topology.

2 Topological Data Analysis in a Nutshell

Firstly, we think of the data existing in an n dimensional Euclidean space, known as the point cloud. Depending upon the type of data, the dimension could vary. In this report we have used point clouds of dimension 756 and 256.

• To use topology on discrete datasets, we use the concept of **Vietoris - Rips complex**. These are created by defining a sphere of radius ϵ around each data point. These spheres with really small can be thought of as 0 simplices. Next, we increase the radius (ϵ) of these spheres and when two spheres intersect, we create an edge connecting the two data points, 1 simplex.

• We create an n dimensional simplex when all its boundaries exist. For instance, a triangle (2-simplex) is formed when ϵ is large enough that all three triangle edges exist.

• To keep track of this, we create a **Distance matrix** (D_{ij}) which collects the Euclidean distances between each individual point. So, an edge between points i and j would occur when $D_{ij} \leq \epsilon$. If we keep increasing ϵ, edges will start to appear.

• Let C_p be the vector space generated by the p-simplices. We define the boundary operator$\partial_p : C_p \to C_{p-1}$, which maps a simplex to its boundary. The p-th homology group is defined as the quotient space:

$$H_p = \frac{\ker(\partial_p)}{\text{im}(\partial_{p+1})} \tag{1}$$

• C_0 are individual points and we define $\partial_0(H_0) = 0$. Therefore, H_0 represents the independent connected components (data clusters). H_1 represents the edges which are not boundaries of a 2 simplex, that is a solid triangle. Therefore, it contains 1-dimensional topological voids (loops of continuous transformation).
To actually apply these concepts, we need to create an algorithm that can track when elements of H_0 and H_1 are formed and when they die. Die in the sense that a bigger complex emerges whose boundary contains them. We use **Boundary Matrix** and **Matrix Reduction** to carry out this task.

3 The Boundary Matrix

(This section is AI generated because sadly it is more efficient)

The boundary matrix is a fundamental algebraic data structure that encodes how these geometric building

blocks connect to one another.

- Structure: It is a square matrix where both the rows and columns represent the simplices of the complex. Crucially, these simplices must be ordered by the time they appear in the data (their filtration value).
- Entries: When computing homology with coefficients in $\mathbb{Z}_2$ (modulo 2 arithmetic), the matrix entry at row i and column j is 1 if the i-th simplex is a direct face on the boundary of the j-th simplex (for example, if vertex i is an endpoint of edge j). Otherwise, the entry is 0.
- Purpose: It translates the geometric structure of the data into a purely algebraic format that can be processed programmatically.

Matrix Reduction (Standard Algorithm)

(This section is AI generated because sadly it is more efficient)

To extract the persistent homology—meaning the "birth" and "death" of topological features like connected components, loops, and voids—we apply a specialised form of Gaussian elimination to the boundary matrix, known as matrix reduction.

- The Pivot: For any given column, the "pivot" (often called the low function) is the row index of the lowest non-zero entry in that column.
- The Process: We iterate through the columns from left to right. If the current column j has the same pivot row as an earlier column k (where $k < j$), we add column k to column j. Because we are operating in $\mathbb{Z}_2$, this is equivalent to an XOR operation ($1+1 = 0$), which cancels out the lowest 1. We repeat this addition until column j either becomes a column of all zeros, or it has a unique pivot row not shared by any preceding column.
- The Output: The reduced matrix directly yields the topological lifecycle of your data (the barcodes). If column j ends up with a pivot at row i, it signifies that the topological feature created by simplex i was "killed" (filled in) by simplex j. If a column becomes completely empty (all zeros), the corresponding simplex gave "birth" to a feature.

Pictorial Example

In this section, we try to give an example using hand-drawn diagrams; please bear with some bad drawings.

Figure 1 shows how the boundary matrix gets filled and how matrix reduction takes place. Firstly, observe that

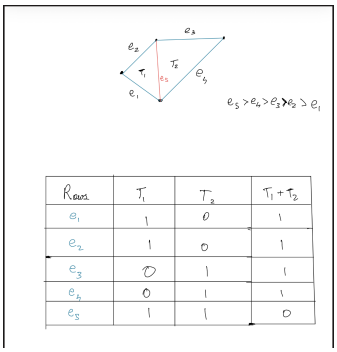

Figure 1: Boundary matrix with two triangles

$e_5 > e_4 > e_3 > e_2 > e_1$ (this is not pictorially accurate as the author is not good at drawing). This means that as epsilon increases, e_1 appears first and e_5 appears last. Accordingly, we fill the boundary matrix with T_1 first, getting 1 at e_1, and subsequently at e_2, e_5. Similarly, we fill the matrix for T_2. According to the matrix reduction algorithm, as T_1 and T_2 both share an edge at e_5, T_2 is converted to $T_1 + T_2$, where $+$ is an XOR operation.

The column $T_1 + T_2$ represents a loop $\in H_1$, and it gets matched with the row having the lowest 1 in the column, i.e., with e_4. This shows that the loop is born when e_4 is born.

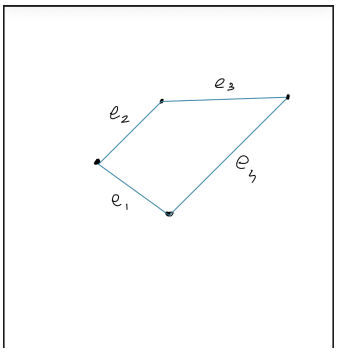

Figure 2: Loop created by edges

This loop dies, or becomes a boundary of a triangle, when ϵ is big enough for e_5 to exist. Thus, when T_2 or e_5 appear, the loop dies. So the birth of the loop created by $e_1 + e_2 + e_3 + e_4$ is when ϵ allows e_4, while it dies when ϵ allows e_5 (fig 7). And the solid lid that closes this loop is T_2, so they are matched accordingly in our algorithm.

4 Methodology

We trained a standard classifier on the MNIST dataset. To extract the topological "brain" of the network, we bypassed the final 10-dimensional softmax probability simplex and extracted the activations from the 256-dimensional hidden layer.

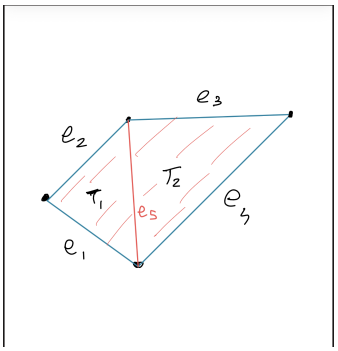

Figure 3: Loop filled by solid triangles

4.1 Neural Network

A diagram of the neural network we trained (AI generated) :

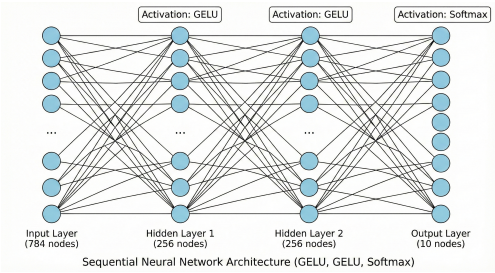

Figure 4: FFNN diagram

Using the raw data will increase the distance between each point as the point cloud would be a 784-dimensional space, while the outputs from the dense layers of this network would only have 256 dimensions.

4.2 Existing library

There is an existing library in Python, Risper, which could be used for topological data analysis. We will use the Risper library to show the output we are expecting from our code.

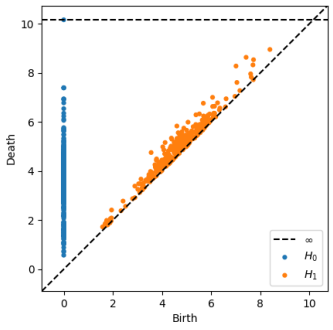

Figure 5: Risper output

We took the output of the last layer of the feed-forward neural network as point cloud data and got it processed in the TDA algorithm. The blue dots here represent the birth and death of H_0, independent connected components. The diagonal represents the structures that are born and die instantly.

5 Results and Visualizations

5.1 Using Raw MNIST data

The raw MNIST data is a set of points in 784-dimensional pixel space, and the distances between distinct images are vast and unorganised.

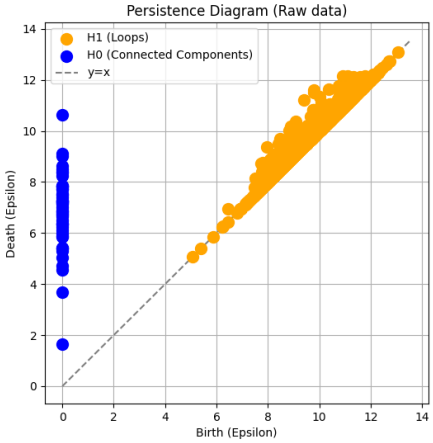

Figure 6: TDA on Raw data

5.1.1 Graph analysis

The vertical axis represents the value of ϵ when a certain complex gets combined with another complex of higher dimension. That is the death value of the complex.

(For example : Two independent points (0-simplex) die when an 1-simplex (edge) appears which connects them.)

The horizontal axis represents the value of ϵ when a certain complex appears in our algorithm. It is the birth value of the complex.

(An edge dies when a 2-simplex appears whose boundary is that particular edge.)

The blue dots in the graph represent the birth and death of H_0, independent connected components. The diagonal represents the structures that are born and die instantly. The orange dots represent the birth and death of 2-simplices.

It can be seen here that the births and deaths occur at large values of ϵ. This is because each data point in the raw MNIST data is a 756 dimensional vector. Hence, the radius to connect each point is large.

The points are very close, which shows that the data is clustered together, unlike the data from the output of a trained network.

5.2 First layer Output

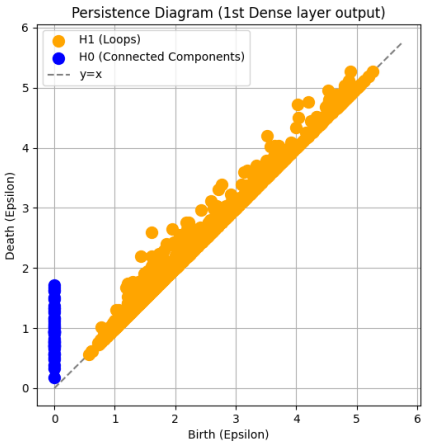

Figure 7: TDA on first layer output

5.2.1 Graph analysis

It is evident that the birth and death rates for this dataset have decreased, yet the points remain clustered. This is attributed to the fact that we extract points from the initial layer of our neural network, which is not accurate.

The reduction happened because now the data passes through our neural network which produces output of dimension 256. Thus, our point cloud is a 256 dimensional space, much less than 756 dimensions.

5.3 Final layer Output

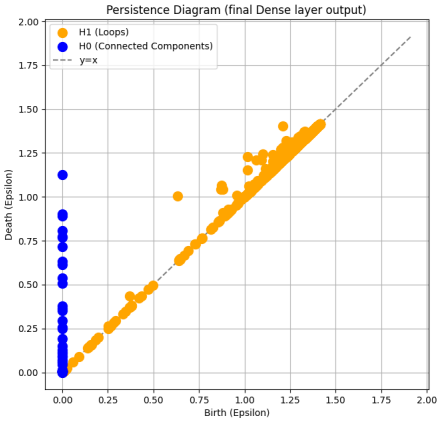

Figure 8: TDA on final layer output

5.3.1 Graph analysis

Finally, we use the output of the final dense layer of our neural network as our point cloud. This is before the data passes through the activator and thus the point cloud is still a 256-dimensional space.

5.4 Observations

- The very clear observation from Figure 8 is that we are finally getting 9 distinct blue dots starting from $\epsilon = 0.5$. This shows that our model has successfully separated the image data points into 10 different islands that can only be connected when the radius value is greater than 0.5. Note : We obtain 9 dots instead of 10 because the final one engulfs all the other spheres and never dies.

- The next observation is the blue dot with the highest death on the graph represents the death of an H_0 simplex caused by the edge connecting the points of Image 27 and Image 37, which are the following :

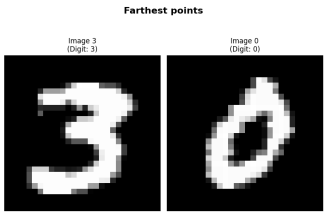

Figure 9: Farthest points

This means that our neural network placed these two points farthest from each other, which makes sense as transforming a 3 into a 0 would take a long chain of continuous steps.

- Next, we will check the most persistent hole H_1, represented by the orange point that dies at $\epsilon = 1$. This is the most persistent loop as it has the maximum vertical distance from the diagonal. Figure 10 shows the endpoints of the edge which created this loop and figure 11 shows the images of the triangle (2-simplex) that ultimately kills the loop.

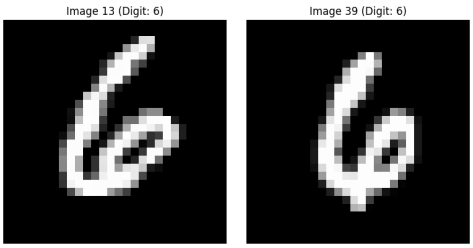

Figure 10: Persistent loop birth

This shows that a loop containing 9s and 6s took the longest time to be filled by a higher simplex. And it only gets filled by a triangle which connects one of the 9s to the 6. This implies that the neural network did not put any

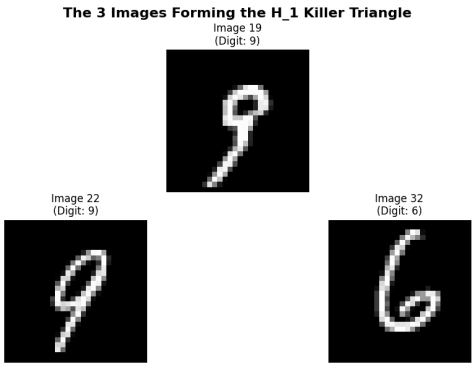

Figure 11: The maximum persistence H_1 loop. The network physically separated the 9, 9, and 6 clusters by routing them in a continuous ring around a central void of geometric ambiguity.

image inside this loop, it was a no man's land, because if there were any data points inside the loop then the hole would be filled sooner.
This makes sense because the pathway a neural network needs to morph the digit 9 into the digit 6 has no intermediate step that might look like any other figure from the dataset. Intermediate steps would be fuzzy scribbles which the MNIST dataset does not include, hence the empty void in the middle.

6 Future Work

6.1 Cosmological Topology

The methodology developed here—tracking the topological evolution of distinct point clusters and continuous voids—is immediately transferable to astrophysics. The large-scale structure of the universe is fundamentally geometric. By replacing neural network activations with cosmological simulations or weak lensing data, we can apply Geometric Deep Learning and TDA to quantify the formation of dark matter halos (H_0), cosmic filaments (H_1), and massive cosmic voids (H_2).

6.2 Non Euclidean Space

This report was carried out by assuming that the space of data, point cloud, follows Euclidean metric. However, not all datasets need to follow that and thus we can improve our understanding by integrating geometry into the algorithm. An important step for analysing cosmological data where the metric is curved. An integration of geometric deep learning (GDL) and TDA would be the next step for the author.