REPORT5: TOPOLOGY OF DEEP NEURAL NETWORKS(PAPER REVIEW)

Brandon D. Finley
Applied Mathematics
University of Colorado, Boulder
Email: brfi3983@colorado.edu

November 2, 2020

1 Introduction

This report will go over **Topology of Deep Neural Networks**, by Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. It will also go over various discoveries I found along the way while I was researching this topic. In general they used tools to evaluate how the topology of the input data changes as it passes through a deep neural network. To do this, they used Betti numbers, Pesistent Homology, and Simplical Complexes to evaluate the topology at each given layer given various architectures. Additionally, they also corroborated their findings from simulated data with real-world datasets.

2 Theoretical Background

In this section, I will go over the background necessary to understand why this paper used this particular method. In particular, I would like to note that although the concepts seem mathematically simple, I had to do much research into how they related to neural networks. Thus, I learned a lot of implicit ideas regarding the manifolds of your data, how the neural network transforms its layer, and how to visualize or confirm your findings through topological data analysis. I hope you also see the value in this paper for someone who is interested in the general reasoning on how deep neural networks work.

2.1 Betti Numbers

Betti numbers are used to describe the topology of a space. In general, homology counts the number of "holes." Now, a 2D hole could simply be a circle lacking space. If you extend this to 3D, it becomes a void. That is, think of a hollow sphere. Although we can not visualize this, if you extend it to n-dimensions, we repeat the same process.

In general, given a topological space $M \subseteq \mathbb{R}^d$, we can say $\beta_0(M)$ counts the number of connection components in M and $\beta_k(M), k \geq 1$ counts the number of k-dimensional holes in M. And so, we can define the set

$$\beta(M) = \{\beta_0(M), \beta_1(M), \ \cdots, \beta_d(M)\} \tag{1}$$

to be the overall topological description of a space $M \subseteq \mathbb{R}^d$.

As this is a set and not a scaler, we can define a metric, $w(M)$, that captures the **topological complexity** of the space, to be the sum of all the elements in $\beta(M)$. Formally we say

$$w(M) := \beta_0(M) + \beta_1(M) + \cdots + \beta_d(M) \tag{2}$$

Notice, the higher the number of Betti numbers, the more "complex" the topology of the space is. Thus, a common theme in this paper is to reduce the complexity of the topological space so that our network can separate the different classes easily. We will be monitoring this metric with different *activation functions, network widths, and network depths.* More importantly, this will be evaluated for each layer, $v_k(M), k = 1, \cdots, l-1$ the data is passed through and so we track

$$\beta(M) \to \beta_{v_1}(M) \to \beta_{v_2}(M) \to \ \cdots \to \beta_{v_l}(M) \tag{3}$$

However, in the world of modern deep learning, problems arise with using high-dimensional data with unknown topology. As you will see, we will be estimating the Betti numbers by sampling from a point cloud and using persistent homology to estimate the topology of our original space M.

2.2 Simplical Complexes

Simplical Complexes will play an extremely important role in this paper as it allows us to construct a geometric

or abstract interpretation from our point cloud so that we can determine the topology as it passes through our network. In short, it uses the concept of Simplex and Faces to create a way to turn a point cloud into something can use homology upon it. Formally, we have written down the definitions along with an illustration below:

Simplex: A k-dimensional simplex, or k-simplex, σ in $\mathbb{R}^d$ is the convex hull of $k+1$ affinely independent points $v_0, v_1, \cdots, v_k \in \mathbb{R}^d$, where $\sigma = [v_0, v_1, \cdots, v_k]$.

Fig. 1: Simplices

Faces(σ): Simplices of dimensions 0 to $k-1$ formed by convex hulls of proper subsets of its vertex set $\{v_0, v_1, \cdots, v_k\}$.

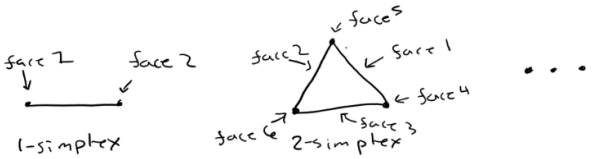

Fig. 2: Faces

Simplical Complex: A m-dimensional geometrical complex K in $\mathbb{R}^d$ to be a finite collection of simplices in $\mathbb{R}^d$ of dimensions at most m that requires that

- the intersection of any two simplices is a shared face
- every face from a simplex within K is also in K

Below, we have a quick illustration of how to apply the definition above.

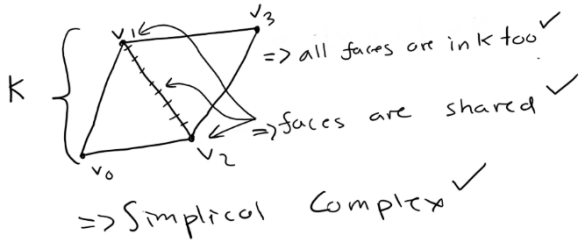

Fig. 3: Simplical Complexes

2.3 Persistent Homology

As we already know that homology is the study of holes in your topological space, we delve into the meaning behind persistence. In short, when we have a point cloud, in order to construct the Vietoris–Rips complex, we must choose a radius for each point ε s.t. if two points' radii overlap, they are then connected as a simplex. If there are multiple overlaps, a higher-dimensional simplex forms. This then creates a simplical complex.

However, for a given dataset, we do not know the specific scale ε to choose so that it captures the topology. Thus, we have the notion of persistent homology where instead of considering a specific ε, you consider *all* potential ε. And so, you keep track of when holes form and when they disappear as you vary ε. The length of the hole's existence to its termination is its *persistence*. Thus, when you consider multiple values of ε and keep track of the persistence for an i^{th} homology, we get a persistence barcode. This is illustrated below from a YouTube video I watched:

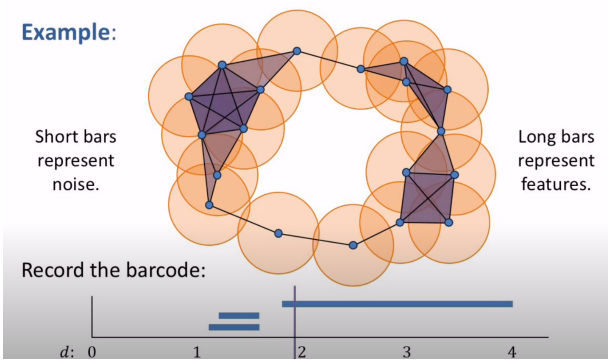

Fig. 4: Persistent Barcode

3 Results

3.1 Simulated vs Realistic Data

The *main issue with using this process to analyze neural networks given a certain dataset is the computational task.* That is, real-world datasets have an unknown topology, and thus requires exploration of a scale for ε to be found via persistent homology. And so, computational needs are dramatically increased for real-world datasets.

Additionally, with simulated data, we have clean data that does not need de-noising while realistic data requires this step. Lastly, this paper was done for "well-trained" neural networks which is unrealistic sometimes as the dataset may be sufficiently complicated and have a generalization error greater than what this method allows.

Although our intermediate layers are in high dimensions, that is the number of neurons, we can try to visualize what is going on by projecting our space onto the first two principal component axis via PCA.

Note: DII and DIII were datasets generated in the paper. DII had multiple interlocking tori and DIII had enclosed sphere within eachother.

3.2 Simulated Data

3.2.1 Activation Functions

For simulated data, we start with activation functions. Comparing the different activation functions, namely ReLU, Leaky ReLU, and hyperbolic tangent, there are some interesting patterns.

Recall that a homeomorphism is a mapping that is 1) continuous, 2) bijective, and 3) has a continuous inverse function. And so, considering hyperbolic tangent, we say

$$\begin{aligned}\tanh(x) &= \frac{\sinh(x)}{\cosh(x)} \\ &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ &\implies \text{continuous since } e^x \text{ is continous}\end{aligned}$$

It is also bijective as each $f(x)$ has a specific, unique input x. Lastly, its inverse function is given by a composition of continuous functions (natural log) and so is also continuous. Now we consider ReLU. If we let $x_1, x_2 \in D$, where the function is defined to be $g : D \to \mathbb{R}$, $x \mapsto \max(Ax + b, 0)$, s.t. $x_1, x_2 < 0$ and $A, b > 0$, then $x_1, x_2 \mapsto 0$. Thus, ReLU is not one-to-one and so is not bijective. Thus, it is not a homeomorphism.

However, the paper shows that ReLU actually simplifies topology a lot faster. This gives motivation to why you might not want to preserve the topological structure of the dataset. This also hints on the previous idea that ReLU is a better problem due to the vanishing/exploading gradient. However, this shines light on another possibility. A similar thing happens with Leaky ReLU as it also simplifies the topology of our data but not as fast as a regular ReLU.

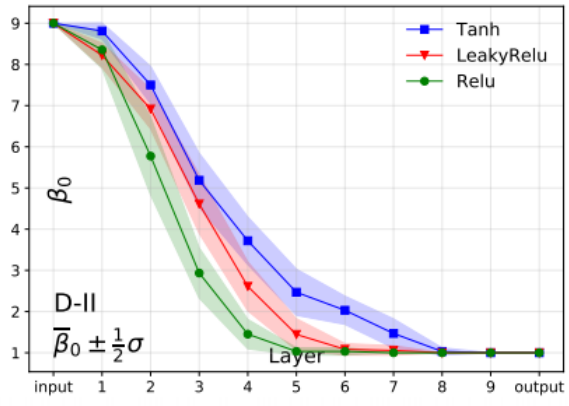

Fig. 5: Lower Complexity

Additionally, note that more complicated topological features cause our network to require more operations, that is layers. This is exacerbated when we use hyperbolic tangent as it tries to preserve its structure and so it takes longer to disentangle our input data into linearly separable manifolds.

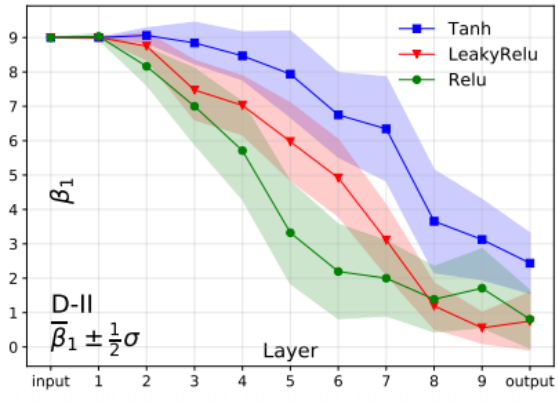

Fig. 6: Higher Complexity

3.2.2 Width

The network also showed that when considering narrow, wide, and bottleneck architectures, topological changes progressed faster on narrower and bottleneck networks. Both of these regimes, I believe, are connected as a narrower network requires more drastic simplification of the topology in order for it to be linearly separable. In the case of the bottleneck layer, it appears that the topology is forced to change due to the low dimensionality of the layer. Below, we have the graph(s) that the paper produced.

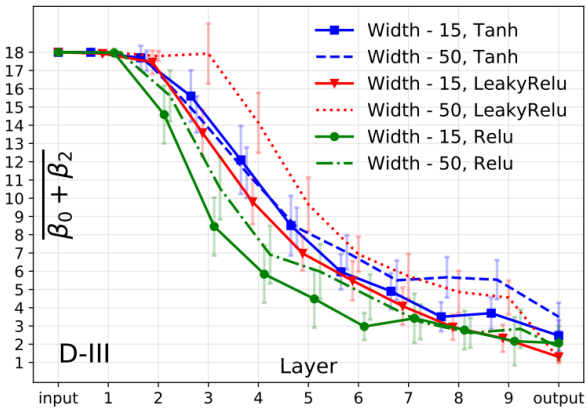

Fig. 7: Topology by Width

3.2.3 Depth

As noted above, depth of the layers is related to the "sequence" of operations needed to simplify the topology of the dataset. Thus, a deeper network allows for more transformations to take place and therefore yields typically more accurate solutions. It was also noticed that as depth decreased, topological changes were not spread our

uniformally across all the layers but were concentrated in the final layers as a drastic attempt to make the data linearly separable. This loss of potential information helps explain why a shallower network might do worse for testing as it makes large changes to the training data in a last effort. Below, we have the graph(s) that the paper produced.

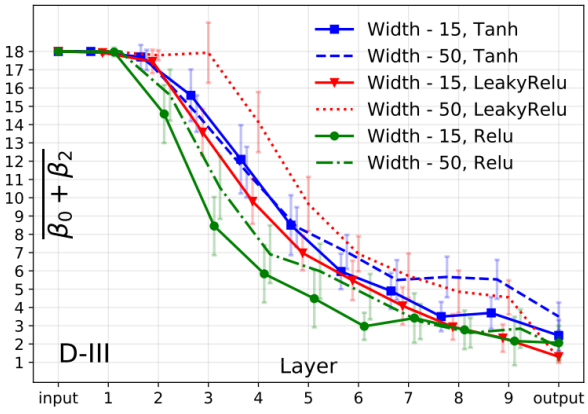

Fig. 8: Topology by Depth

3.3 Real Data: MNIST

3.3.1 Confirming Results

As MNIST is known to be trained to high accuracy, it was selected as one of the datasets to confirm the findings in the paper. However, recall that for our method to work, we need a point cloud data source. Thus, we change our the resolution of our training images' resolution from 28×28 to a vector in 784 dimensions. That is, we flatten our image and change it to a vector, say $x \in \mathbb{R}^{784}$. As this is a high dimensional vector, but at the same time, one of the lowest resolution datasets for images available, at least compared to CIFAR-10 and ImageNet, we must reduce its dimensionality for computional purposes. Using PCA, the resolution is decreased to 50 dimensions, that is now $X \subseteq M \subseteq \mathbb{R}^{50}$. In the paper, it is shown that the images retained most features.

3.3.2 Computational Tricks

As mentioned above, since our dataset, MNIST, has an unknown topology, we need to be wary of the computational tasks persistent homology poses and so we consider a finite set of possible ε values. In the end, they used the top 3 values that captured that most meaningful information. In the paper, it mentioned that as we are only considering topology, we can simplify our topological structure that reduced the dimensionality but maintained its topological values. This is a must for real world data as, once again, this method is very taxing. So this begs the question: *what advancements are there for computing topological features on real life datasets?* Or does it not matter if we can confirm it for a particular case, that others follow?

There are many questions that I pulled from this paper and am interesting in for the future.

4 Miscellaneous Discoveries and Future Work

During this paper, the main question that arose was the use of this method. Namely, its computational cost is so high, it would not be very useful to studying datasets. However, it mentioned a paper that for 3×3 greyscale images, it found that the topology of that dataset closely resembled a klein bottle. This idea was then extended to data compression. This reminded me of a hypothesis I came across: **Manifold Hypothesis** which explained that many datasets lie on a low dimensional manifold. Thus, its complexity can be reduced.

This also reminded me of **autoencoders**, as they are forced to learn the structure of the dataset as a bottleneck layer is introduced. Although reading over it briefly, I saw that this is the reason why autoencoders and manifold learning is intertwined. Thus, I wonder how we can use autoencoders to assist in learning how our datasets are structured and thus be able to make informed decisions on what type of neural networks (if any) to use on them.

Additionally, when reading about the point cloud being created from the MNIST dataset, I came across the use of embedding for various uses. For instance, using a sliding window for time series data and embedding it into higher dimensions showed its periodicity. Along this this, I stumbbled across various dimensionality reduction techniques such as t-SNE. And while I enjoy the topic(s) that were proposed in this paper, I see the value it lacks and what these other techniques provide. In the end, I hope that we can build a complete picture of all of these topics and their connections.

As such, for the next report, I believe I will explore autoencoders and dimensionality reduction so that I can personally take part in the programming and explore various datasets. I may save this for later, but I do want to look at how the topology changes due to your hyperparamters too.