Winter 2022 Mathematics Capstone: Topological Data Analysis on Well-Trained Neural Networks for Binary Classification.

Adam Goeddeke

April 2022

Abstract

Topological data analysis, or TDA, is the field of study associated with analyzing how data is connected. It begins with the idea that large sets of data have a notion of "shape" that is topologically significant. Here, we are basing our findings off our own conclusions and experiments inspired from an article published in the twenty-first issue of the *Journal of Machine Learning Research* [2] by Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. The data we are using is an open-source repository of images of counterfeit and genuine bank notes that have been quantified using the processes of Wavelet Transformation. This allows us to put our data set in a point cloud which is a subset of $\mathbb{R}^4$. In particular, we investigate the changing topology of data as it passes through the layers of a well-trained neural network. By "well-trained" we mean one whose accuracy is 99% or better. Once our network is trained, we can extract its weights and biases and send the data through it again. We then take the output vectors of each layer and use PCA to reduce the dimension of the point cloud from four to three for better visualization. We will then plot persistence diagrams and barcodes of each point cloud to analyze how the shape of the data changes at each layer.

1 Introduction

In the article presented by Naitzat, Zhitnikov, and Lim [2], they trained a deep neural network and analyzed the changing topology of the point cloud generated by the data set as it passed through each layer of the network. Here, our data consists of 1,372 unique images of banknotes that are classified as either genuine or counterfeit. The raw data consists of grey-scale images ranging in size from 96x96 pixels to 128x128 pixels [2] [3].

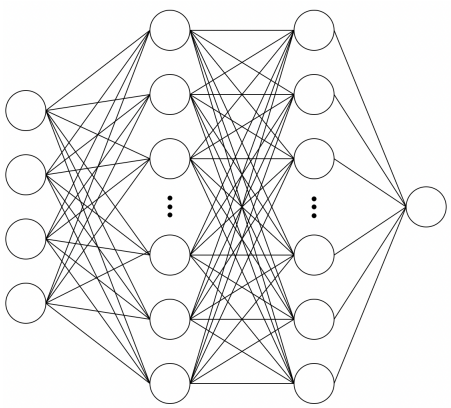

Figure 1: Neural network with 4 input nodes, two hidden layers, and one output node.

Neural networks are sometimes referred to as "black-boxes," meaning that very little is known about the process inside the network as it is being trained. In order to investigate what is happening inside the network, we follow this process:

1. Train the network on our data set.
2. Send the data back through the network extracting the output of each layer it passes through.
3. Plot the output of each layer in $\mathbb{R}^3$ by its principal components.
4. Plot the persistence diagram and persistence barcodes of each layer.

2 Background

2.1 Neural Networks

For some background, we will discuss how a neural network actually functions. Figure 1 is a diagram of the neural network we are using. We will go into more detail about it specifically later. For now, we stick how neural network function on a general level. A standard neural network has three different types of layers:

(i) Input Layer (the first layer pictured)

(ii) Hidden Layer(s) (the middle two layers pictured)

(iii) Output Layer (the last layer pictured)

The input layer, as its name would suggest, is the layer where we input our data into the neural network. The hidden layer(s) come next and include all of the

following layers excluding the output layer. Typically, these are fairly large and contain many nodes. For example, if we refer to network A being "deeper" than network B, all we mean to say is that network A contains more hidden layers than network B. Finally, the output layer is similar to the input layer in that it is the layer which contains the output of our network.

There are two basic processes in a neural network. The first being "feed forward" and the second being "back propagation." We shall describe the feed forward process in detail, as this is the main focus of this paper, but only briefly cover back propagation since it is less important for our purposes. The TensorFlow library does both of these processes for us with very little code on our part.

To feed forward data in a neural network is to send a single datum through the neural network once. We first bring to your attention weights, biases, and activation functions. Weights and biases are column vectors and real numbers, respectively, stored in each node of the hidden layer(s) and output layer. These weights and biases are what get altered during the back propagation process to try to make the next feed forward iteration produce a more accurate output.

Activation functions define the output of a specific node. We will be working with three main types of activation functions: Rectified Linear Unit (ReLU), Hyperbolic Tangent (tanh), and Sigmoid. The ReLU and tanh activation functions are very useful in many deep neural network applications and are typically used in the hidden layers. The sigmoid activation function is very common in the output layer of binary classification problems.

$$\text{Sigmoid: } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\text{ReLU: } R(z) = max(0, z)$$

$$\text{tanh: } tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

When training a neural network we need a target value to compare our output, $\hat{y}$, with we will call this y. To feed forward, we take our input layer with, say, $i \geq 1$ nodes. Then, the data we are inputting should be a column vector with i rows which we will call $\overrightarrow{x}$. We take each $x_i \in \overrightarrow{x}$ and "send" it to each node in the first hidden layer. It should be noted that the input layer does not have an activation function or weights and biases; it simply acts as a conduit to start the feed forward process. Let's suppose that in our first hidden layer we have $h_1 \geq 1$ nodes. In general, we will say that each hidden layer has $h_n \geq 1$ nodes where n is the number of hidden layers (that is, h_1 corresponds to the first hidden layer, h_2 to the second, and so on). We then multiply the transpose of our weight vector by the input and add the bias. In symbols, we write

$$z_i^{[n]} = (\overrightarrow{w_i^{[n]}})^T \cdot \overrightarrow{a} + b_i^{[n]}, \quad \forall i \in \{1, 2, ..., h_n\}.$$

Where $\overrightarrow{w}$ is the weight vector, $\overrightarrow{a}$ is the output vector of the previous layer (in the case of the first hidden layer, $\overrightarrow{a} = \overrightarrow{x}$), and b is the bias. Additionally, we use the notation $z_b^{[a]}$ to denote the output of the b'th node in the a'th layer *before* applying the activation function. Once we have calculated each $z_i^{[n]}$, we apply its layers' respective activation function and obtain the output of the current layer which is the input of the next layer. We will call this $\overrightarrow{a}'$.

We iterate this process for each layer in our neural network. Once we get the output for the output layer (traditionally, $\hat{y}$ is used to denote the output vector of the output layer over the previously used $\overrightarrow{a}$ notation) we compare it to the expected value y. We then calculate the discrepancy between our $\hat{y}$ and y using a loss function. Finally, we proceed with the back propagation process to tweak the weights and biases to allow for a more accurate $\hat{y}$ prediction. This gets repeated many times until we find a combination of weights and biases that most accurately predict the expected y value.

2.2 Persistent Homology

Persistent Homology is the process of computing the topological features of a point cloud in a given space. In order to understand persistent homology, we first need to understand the idea of a simplicial complex. We can then apply a filtration to that complex and visualize it using persistence diagrams and barcodes. These persistence diagrams and persistence barcodes will be especially usefully later when we analyze the topological complexity of the point clouds generated by the output vectors of each layer in the neural network.

Definition 2.1 *[1] For $k \geq 0$, a k-simplex, σ, in an Euclidean space $\mathbb{R}^n$ is the convex hull of a set P or $k+1$ affinely independent points in $\mathbb{R}$. In particular, a 0-simplex is a vertex, a 1-simplex is an edge, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. A k-simplex is said to have dimension k.*

Definition 2.2 *[1] A simplicial complex $\mathcal{K}$ is a set of simplices that satisfies the following conditions:*

(i) Every face of a simplex of $\mathcal{K}$ is also in $\mathcal{K}$.

(ii) The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.

The simplicial complex is essentially a grouping of k-dimensional simplicies in $\mathbb{R}^n$. By understanding how these simplicies are connected we can compute their homology groups and, by extension, the homology groups of any structure homotopy equivalent to the simplicial complex.

Definition 2.3 *[1] Let (P, d) be a finite metric space. Given a real $r > 0$, the Rips Complex is the abstract simplicial complex $\mathbb{VR}^r(P)$ where a simplex $\sigma \in \mathbb{VR}^r(P)$ if and only if $d(p, q) \leq 2r$ for every pair of vertices of σ.*

The Rips Complex allows us to quickly compute the homology groups of finite point clouds. Without the Rips Complex, we would be unable to draw any topological significance from a finite point cloud because it is just that, a finite collection of points with nothing connecting one point to another. The Rips Complex places a closed ball with radius r centered at each point in the point cloud. Varying the value of r allows us to extract different topological significance. Any overlap in these closed balls denotes a simplex between their respective points. The dimension of this simplex is $b-1$ where b is the number of closed balls intersecting. We then compute the homology group of the simplicial complex and call it the homology group of the Rips Complex on (P,d).

The birth and death of a simplex is going to play a large role in this investigation. As the radius of our closed balls grows, we begin to see various topological structures form that are relevant to our discussion. In order to properly visualize this process, we will be using the open source library Gudhi. Specifically, we will be using the persistence diagram and persistence barcode functionality that Gudhi offers.

Definition 2.4 *[1] Let K be a finite abstract simplicial complex. For integers $q \geq -1$, let $C_q(K) :=$ The free $\mathbb{Z}_2$-vector space on the set of q-simplicies of K. For integers $q \geq 0$, let*

$$\partial_q : C_q(K) \to C_{q-1}(K)$$

be the $\mathbb{Z}_2$-linear map defined by linearly extending this assignment on the basis elements

$$\partial_q(\{v_0, ..., v_q\}) = \sum_{i=0}^{q} \{v_0, ..., \hat{v_i}, ..., v_q\}.$$

Where $\hat{v_i}$ means to leave out v_i. Then, consider the linear maps

$$C_{p+1}(K) \xrightarrow{\partial_{p+1}} C_p(K) \xrightarrow{\partial_p} C_{p-1}(K).$$

We say that the $\underline{p^{th} \text{ homology group of } K}$ is

$$H_p(K) := \frac{ker(\partial_p)}{im(\partial_{p+1})}.$$

Where K is a finite abstract simplicial complex.

2.3 Motivation

As an example, how can we topologically describe an empty triangle? The empty triangle is the realization of an abstract simplicial complex with three zero-simplicies and three one-simplicies. The empty triangle does not have any two-simplicies as it is empty. The simplicial complex is

$$K_t = \{\{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}\}$$

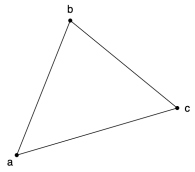

Figure 2: Empty triangle with Vertices a, b, and c.

Where $\{a\}$, $\{b\}$, and $\{c\}$ are the vertices of the the triangle, and $\{a, b\}$, $\{a, c\}$, and $\{b, c\}$ are the edges. Figure 2 shows this realization. Thus, the homology group of the empty triangle, K_t, is

$$H_p(K_t) \cong \begin{cases} \mathbb{Z}_2 & \text{if} \quad p = 0 \\ \mathbb{Z}_2 & \text{if} \quad p = 1 \\ \{0\} & \text{if} \quad p \geq 2 \end{cases}.$$

Theorem 1 *Let S^n denote the empty, closed sphere in $\mathbb{R}^{n+1}$. That is,*

$$S^n = \{(x_1, x_2, ..., x_{n+1}) \in \mathbb{R}^{n+1} \mid x_1^2 + x_2^2 + \cdots + x_{n+1}^2 = r\}$$

for some radius $r > 0$. Then,

$$H_p(S^n) \cong \begin{cases} \mathbb{Z}_2 & \textit{if} \quad p = n \\ \{0\} & \textit{if} \quad p \neq n \end{cases}.$$

From Theorem 1, we can gather that $H_p(S^n) \cong H_p(K_t)$ when $n = 1$. This is because the empty triangle is homeomorphic to the the empty circle. So, too, is the empty sphere homeomorphic to the empty tetrahedron. If we Let S^2 be the empty, closed sphere in $\mathbb{R}^3$. That is, for some radius $r > 0$,

$$S^2 = \{(x, y, z) \in \mathbb{R}^3 \mid x^2 + y^2 + z^2 = r\}.$$

Since $S^2 \cong K_T$, where K_T is the empty tetrahedron, it can be be shown that

$$ker(\partial_2) = \mathbb{Z}_2$$

$$im(\partial_3) = \{0\}.$$

Thus, we have the quotient

$$H_2(S^2) = \frac{ker(\partial_2)}{im(\partial_3)} = \frac{\mathbb{Z}_2}{\{0\}} = \mathbb{Z}_2.$$

Its dimension, $dim(H_2(S^2)) = 1$, is the number of 3-dimensional voids in S^2. The p^{th} homology group of a simplicial complex is very important to our discussion. We will refer to what are known as "Betti numbers" which are just the dimensions of homology groups of simplicial complexes. In general

$dim(H_0(K)) = \beta_0 :=$ The number of path-connected components of K

$dim(H_1(K)) = \beta_1 :=$ The number of unique loops that are not boundaries of K

$dim(H_2(K)) = \beta_2 :=$ The number of three-dimensional voids in K.

In our above example, we would say that the third Betti number of S^2 is one. Because, as we can see geometrically and can be shown algebraically, the number of three-dimensional voids in a empty sphere and an empty tetrahedron is one.

3 Data Set Used and Exploratory Data Analysis

For this investigation, we use the open-source data repository published by the University of California-Irvine (UCI) on their website. Specifically, we are using the "Banknote Authentication Data Set," [3]. The data set has the following schema:

Attributes	Data Type
Variance of Wavelet Transformed Image	float
Skewness of Wavelet Transformed Image	float
Curtosis of Wavelet Transformed Image	float
Entropy of Image	float
Class	int

These are the attributes of the images taken of counterfeit and genuine banknotes after they were quantified using a wavelet transformation process. More information can be found about what this process specifically *does* to the images, but for our purposes we need only know that these five attributes hold information about the images of these banknotes.

The first four attributes (Variance, Skewness, Curtosis, and Entropy) hold information about the image itself while the last attribute (Class) denotes whether or not this specific banknote is genuine (denoted by a 1) or counterfeit (denoted with a 0). This is our y-value while $\overrightarrow{x}$ contains the variance, skewness, curtosis, and entropy values. This data set contains the aforementioned attributes for 1,372 unique banknotes. However, as it is posted on UCI's repository at the time of writing, there are more counterfeit banknotes in this data set than there are genuine. We see this is figure 3.

In order to prevent a bias from being trained into our network, we must first ensure that we have the same number of counterfeit and genuine banknotes. As there are 610 genuine banknotes and 762 counterfeit, we randomly delete 152 counterfeit notes. After this, we have 1,220 (610 counterfeit and 610 genuine) unique banknotes that we will train and test our neural network with.

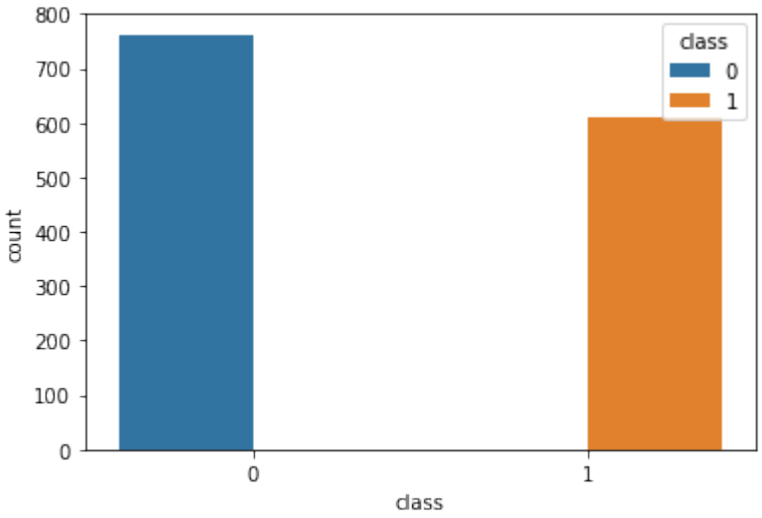

Figure 3: Number of Data Points Labeled Counterfeit (Blue) and Genuine (Orange)

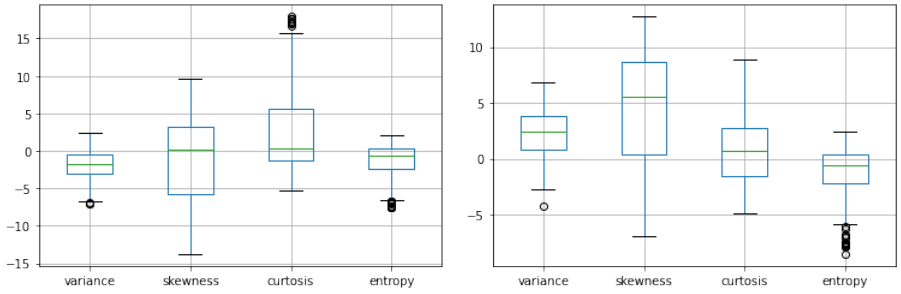

Figure 4: Box Plots of Genuine (Left) and Counterfeit (Right) Banknotes by attribute.

We now look at how the data is distributed by attribute for each class. We can see the distribution of values in figure 4. The distribution for genuine and counterfeit bills are relatively similar. The only main discrepancy that is noticeable is that there are more outliers in the entropy and curtosis for the counterfeit and genuine bills, respectively. It should be noted that the values for the genuine banknotes are more spread out when compared to values of the counterfeit notes. This shouldn't make too big of an impact in our investigation, as these are the types of problems that neural networks are very good and solving. That is, classifying values into groups when the values look very similar to the human eye.

4 Neural Network Construction

To train our neural network, we opted to use the open-source libraries TensorFlow and SciKit-Learn. We utilize SciKit-Learn to split up the training and testing data sets into the standard 80-20 train-test split. We then use TensorFlow to construct the neural network and train it. The general architecture we choose is a fairly simply 4 x 20 x 20 x 1 network. That is, an input layer with 4 nodes (one for each attribute), two hidden layers with 20 nodes each, and an output layer with one node for the classification.

As this is a binary classification problem, the activation function, optimizer, and loss functions are fairly standard. We utilize a Rectified Linear Unit (ReLU) or Hyperbolic Tangent (tanh) activation function on our hidden layers and a Sigmoid activation function on the output layer. We then use the Adam optimizer and a Binary Cross Entropy loss function. After training for 60 epochs, we have a well-trained neural network with an accuracy of 100%.

5 Topological Data Analysis

In order to analyze how the shape of our data changes as it passes through each layer, we need to extract the output vectors from each layer. Unfortunately, TensorFlow does not have this functionality on the scale that we need. So we create out own loop that mimics the feed-forward process using the weights and biases from the well-trained network. We send our entire data set back through this loop and, after each layer, we extract the output vector $\overrightarrow{a}$.

Once we have the output vector for each layer, we can apply Principal Component Analysis (PCA) to plot the 4-dimensional point cloud in three dimensions. We then color each point based on its class. In figures 5-8, we use the colors blue and red to denote counterfeit and genuine banknotes, respectively.

As we can see, the separation in the data is not as prominent until the final layer. This is expected as our neural network is rather shallow. If we were

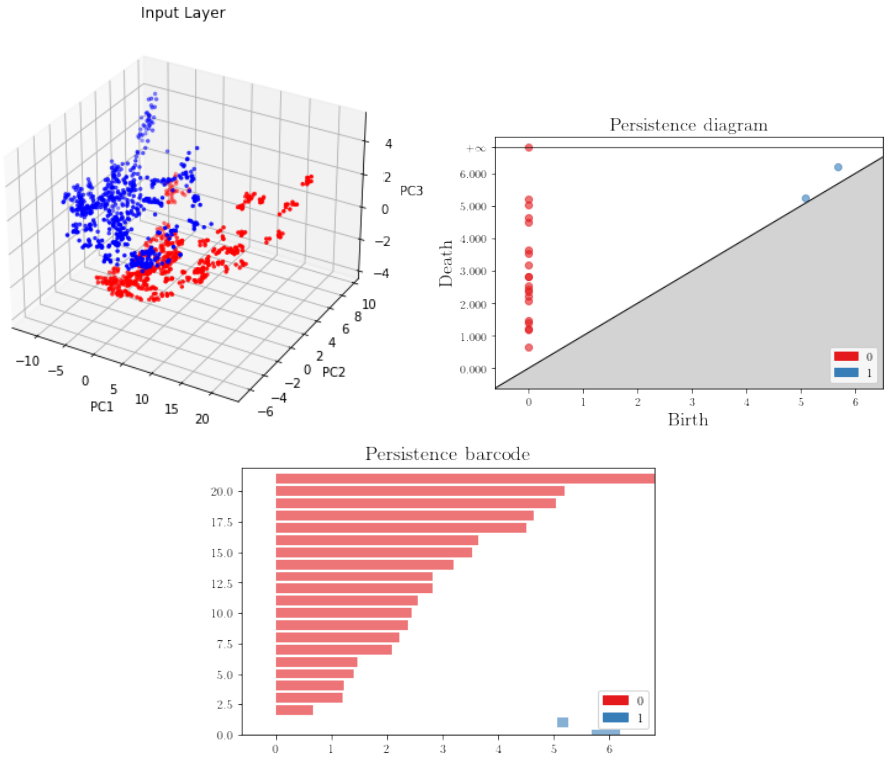

Figure 5: PCA (Left), Persistence Diagram (Middle), and Persistence Barcode (Right) of the Input Layer

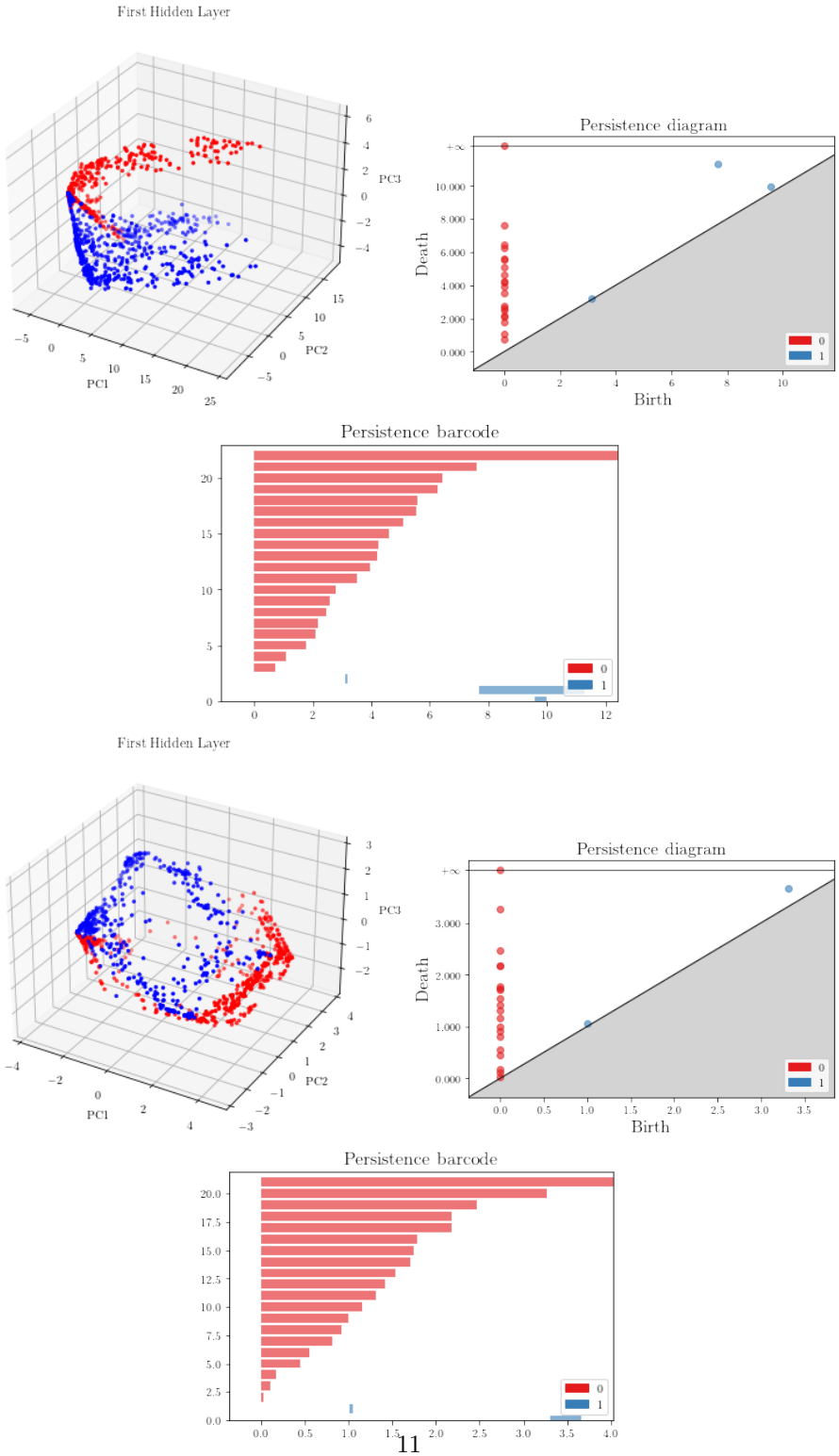

Figure 6: PCA (Left), Persistence Diagram (Middle), and Persistence Barcode (Right) of the First Hidden Layer. Upper Group Corresponds to ReLU, Lower Group Corresponds to tanh.

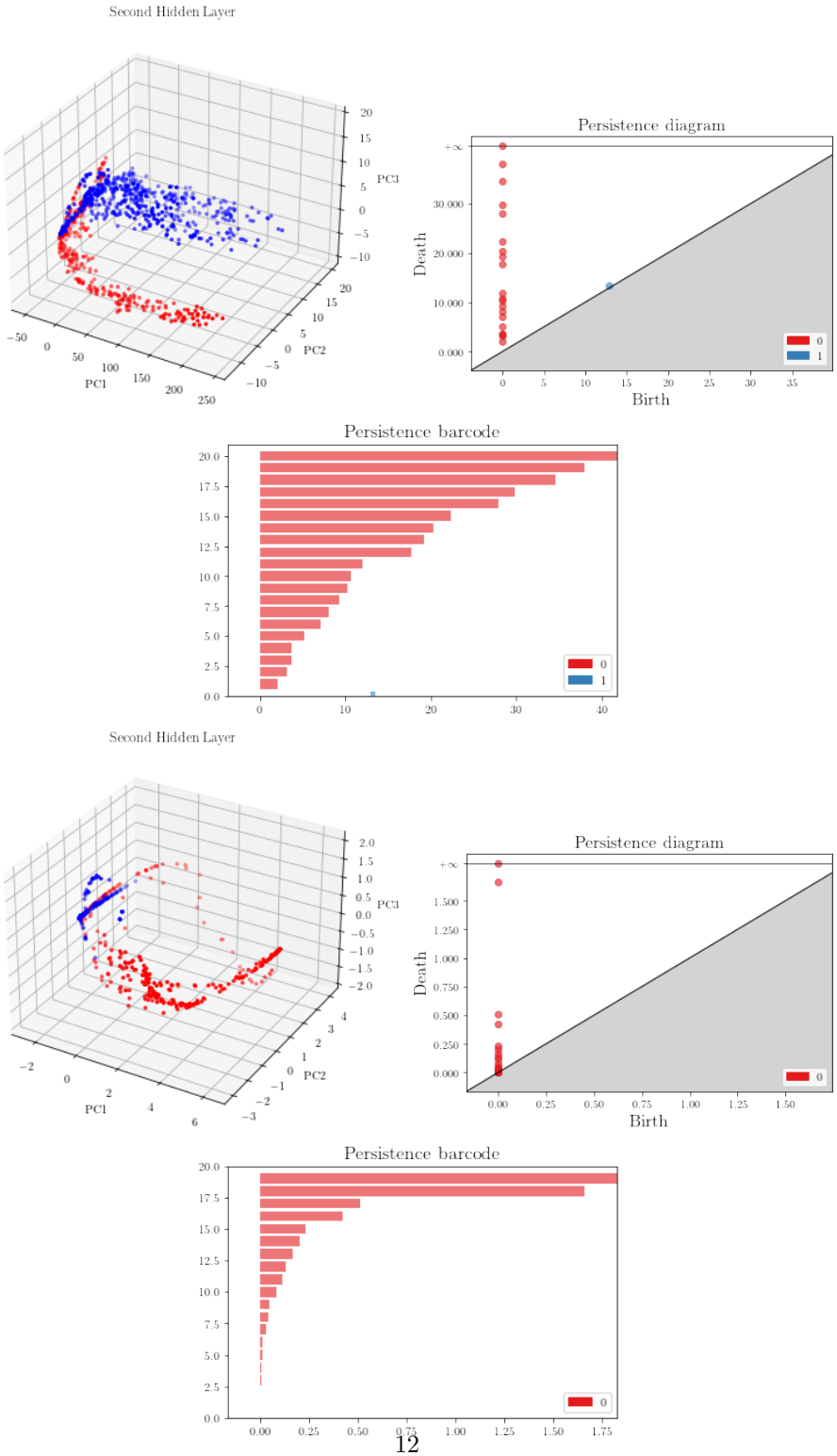

Figure 7: PCA (Left), Persistence Diagram (Middle), and Persistence Barcode (Right) of the Second Hidden Layer. Upper Group Corresponds to ReLU, Lower Group Corresponds to tanh.

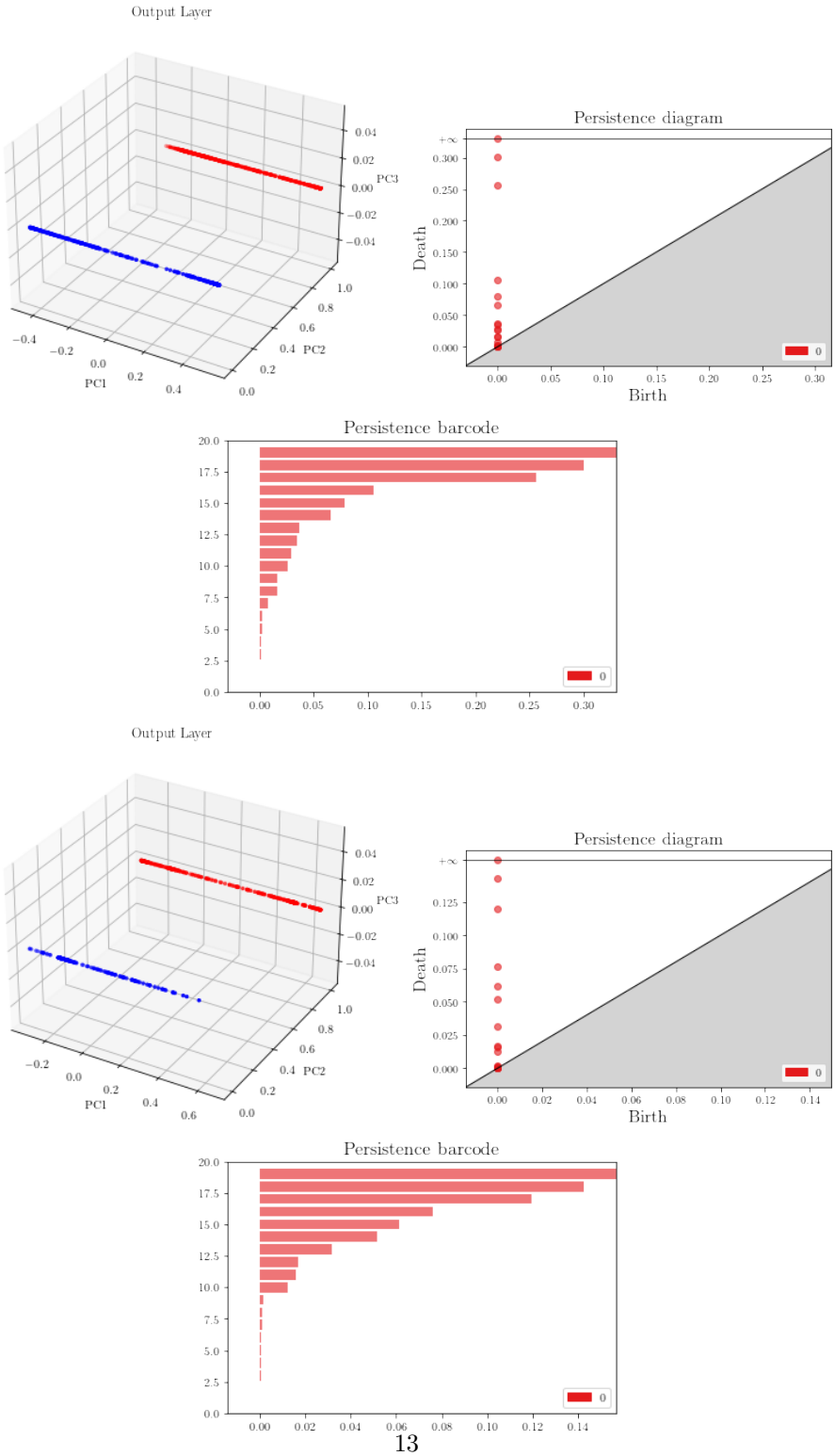

Figure 8: PCA (Left), Persistence Diagram (Middle), and Persistence Barcode (Right) of the Output Layer. Upper Group Corresponds to ReLU, Lower Group Corresponds to tanh.

training a deeper neural network, we would see more complexity and a more gradual process of separation. Typically, a "deep neural network" is classified as a neural network with at least two hidden layers. Our network does indeed meet this qualification. However, to see this separation gradient we speak of, we would need a number of hidden layers more in the range of fifteen to twenty [2]. It should be mentioned that while we would have liked to include a significantly deeper neural network to highlight this fact, the process described in this paper is very computationally expensive. Thus, we stick to more shallow networks here.

The goal of our neural network, from a computational standpoint, is to accurately predict whether a banknote is genuine or counterfeit given its variance, skewness, curtosis, and entropy. Mathematically speaking, what we are doing is reducing the topological complexity of the point clouds generated by this data set. This process allows us to more generally understand what is happening as the data passes through the neural network. That is, we want to reduce the Betti numbers after each layer.

In figures 5-8, we see the output of each layer after passing the data set through it. PCA applied to the data set allows us to visualize it in $\mathbb{R}^3$. We then apply the Rips Complex to each point in the point cloud and plot its homology groups using Gudhi's persistence diagram and persistence barcode functions.

6 Conclusions

What do these persistence diagrams and persistence barcodes tell us about the data and our neural network? It shows us what the neural network is doing to the data, topologically speaking. In general, we can say that the lower the homology group, the more simple the topological structure. That is, a 1-simplex is more "simple" than a 2-simplex. This is what we are seeing as the data passes through our neural network.

The deeper into our network we travel, the more simple the topological structures become. In the persistence barcode for the first hidden layer, we see that the 1-simplex stays alive for quite sometime. Using the ReLU activation function, in the second hidden layer, the 1-simplex is barely even visible. By the output layer, the 1-simplex is completely gone. When we use the tanh activation function, we see the 1-simplex die much earlier on - in the first hidden layer. By the second hidden layer, we no longer have any 1-simplicies. This means that the points of our data set are grouped so tightly together that, when the Rips Complex is applied, there are no 1-simplicies found. This is exactly what we want to do when we train a neural network on a binary classification problem. We are trying to find the simplest way to group data to accurately predict, in our example, whether a bank note is genuine or counterfeit.

We can use this process to more concretely quantify the effectiveness of certain hyper parameters in a neural network; specifically, the activation function. The network with the hyperbolic tangent activation function was significantly faster at reducing the Betti numbers than the ReLU activation function. In fact, from the input layer to the first hidden layer of the model with the ReLU activation function, we actually see an *increase* in the number of simplicies with non-zero dimension.

The faster the model reduces the Betti numbers, the faster our network can predict the data we input into it. Thus, by way of persistent homology on the output of each layer in a deep neural network trained for binary classification, we can conclude that the hyperbolic tangent activation function is more efficient than the rectified linear unit activation function in this example.

What's more, we are actually able to quantify how the data is changing as it passes through each layer. The "shape" of the data, that we mentioned earlier, is becoming more simple after each layer. By "simple" we mean that the Betti numbers are reducing as the data set travels through the network. This is what we would anticipate from a neural network. We would want to see it clustering the data more based on its class the further into the network it goes.

Overall, we were able to successfully verify the results that Naitzat, Zhitnikov, and Lim published on this data set on our specific network. It should be reiterated that in their article the authors explored a much more thorough analysis of the topics we covered here. Due to the computationally expensive processes involved with training neural networks and then using persistent homology to investigate them, we were only able to investigate this on a very shallow level. Ideally, a large variety of network architectures should be utilized.

References

[1] Dey, T. K., Wang, Y. (2022). Computational topology for Data Analysis. Cambridge University Press.

[2] Naitzat, G., Zhitnikov, A., Lim , L.-H. (2020). Topology of Deep Neural Networks. Journal of Machine Learning Research, 21(1).

[3] University of California Irvine (n.d.). UCI Banknote Authentication Repository. UCI Machine Learning Repository: Banknote Authentication Data Set. Retrieved April 14, 2022, from https://archive.ics.uci.edu/ml/datasets/banknote+authentication