# On Algebraic Topology and Neural Networks

**Ricardo Macias**
MATH 6441
Georgia Institute of Technology

## Abstract

Neural Networks have the ability to: diagnose cancers more accurately than an oncologist, beat the world champion of the game "Go", beat professional players in the video game "Dota 2", and represent any continuous function. However, neural networks are difficult to interpret. Computational Algebraic Topology methods can offer insight into the "shape" of the data, and there is interest in the Deep Learning community as to the question of whether these methods can be integrated into network architectures so as to increase how interpretable and expressive these algorithms are. This paper provides a short exposition of two pieces of current research in this sub-field at the intersection of Deep Learning and Algebraic Topology.

## 1 Neural Networks

### 1.1 Background

**Feedforward Neural Networks.** The goal of a feed-forward neural network is to approximate some function $f^*$ with some auxiliary $f_\theta$, where $\theta \in \Omega$ and $\Omega$ is a set of parameters for $f$. In the case of feed-forward neural networks, $f_\theta(x)$, where $x \in X$ is an arbitrary data point, is represented as the repeated composition of functions $f^{(n)}_{\theta_n}(...f^{(2)}_{\theta_2}(f^{(1)}_{\theta_1}(x)))$ each with their own *weights* $\theta_i$. Define the $j^{th}$ *unit* as $a_j = g(\sum_{i=0}^{k} w_{ij}a_i)$ where each $a_i \in \mathbb{R}$ is the output of a unit in the previous layer and each $w_{ij} \in \mathbb{R}$ is a weight associating the units $a_i$ and $a_j$. Essentially, we arrange our network in layers, so that each unit receives input only from units in the immediately preceding layer. $g$, commonly referred to as an *activation function*, must be non-constant, bounded, monotonically increasing, and continuous. Informally speaking, one can consider the union of each $w_{ij}$ "connecting" layers i and j as $\theta_j$ and the union of each $a_j$ at layer $j$ as $f^{(j)}_{\theta_j}(...f^{(2)}_{\theta_2}(f^{(1)}_{\theta_1}(x)))$. For example, in figure 1: Blocks 1 and 2 make up the input or data point $x$, units 3 and 4 make up layer 1 or $f^{(1)}_{\theta_1}(x)$, where $\theta_1$ is made up of the weights "$w_{1,3}$", "$w_{1,4}$", "$w_{2,3}$", "$w_{2,4}$".

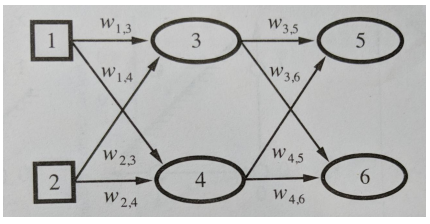



Figure 1: Feedforward Neural Network

**Optimizing Feedforward Neural Networks.** We begin with the assumption that we have access to outputs $Y = \{f^*(x) : x \in X\}$, where $X$ represents our data set. In the machine learning community, the set $\{(x, f^*(x)) : x \in X, f^*(x) \in Y\}$ is known as the *training set*. We consider the best approximation of $f^*$ within the training set to be the choice of $\theta \in \Omega$ such that the error between $f_\theta$ and $f^*$ is minimized. There's many error metrics that can be used, but for pedagogical purposes we show how $f_\theta$ can be "trained" to imitate $f^*$ in $X$ when the error metric of choice is the $L_2$ *loss*. The commonly used $L_2$ loss is defined as $|f^*(x) - f_\theta(x)|^2$, for some $x \in X$.

If we wish to decrease the quantity $|f^*(x) - f_\theta(x)|^2$, then we must come up with a method for adjusting $\theta$ appropriately. The change of the $L_2$ loss with respect to $\theta$ is $\frac{\delta}{\delta\theta}L_2$. Since the gradient is pointing in the direction of steepest ascent and we wish to minimize $L_2$, then we update $\theta$ as follows: $\theta \leftarrow \theta - \alpha\frac{\delta}{\delta\theta}L_2$, where $\alpha$ is known as the *learning rate*. Intuitively, $\alpha$ controls how large the "jumps" we are making down the optimization landscape of $\theta$ are. Using this fundamental idea, we derive how to update the weights of the network. Let $a_k = g(in_k)$, where $in_k$ are all the inputs to $a_k$, be a unit in the last (output) layer in the neural network, $w_{j,k}$ be a weight connecting units $a_k$ and $a_j$, and $y_k$ be the corresponding output in the training set.

$$\begin{aligned}\frac{\delta L_{2_k}}{\delta w_{j,k}} &= 2(y_k - a_k)\frac{\delta g(in_k)}{\delta w_{j,k}} = 2(y_k - a_k)g^{'}(in_k)\frac{\delta in_k}{\delta w_{j,k}} \\ &= 2(y_k - a_k)g^{'}(in_k)\frac{\delta}{\delta w_{j,k}}(\sum_j w_{j,k}a_j) = 2(y_k - a_k)g^{'}(in_k)a_j \\ &\implies w_{j,k} \leftarrow w_{j,k} - \alpha(2(y_k - a_k)g^{'}(in_k)a_j)\end{aligned} \tag{1}$$

The computation of the gradient for weights at earlier layers of the network is similar, as we would repeatedly apply the chain rule based on how early on in the network the weight lies. For instance, if we wanted the gradient for some weight in the layer $j-1$, we would apply the chain rule twice instead of once. The algorithmic procedure that updates the weights in practice is called *backpropagation*.

### 1.2 Theoretical Properties of Neural Networks

**Auxiliary Universal Approximation Theorem.** Let $\sigma$ be a sigmoidal function. Let $I^n$ denote the n-dimensional unit hypercube $[0,1]^n$. the space of continuous functions on $I^n$ is denoted by $C(I^n)$. Let $\epsilon > 0$, $f \in C(I^n)$, $N \in \mathbb{Z}$, $v_i, b_i \in \mathbb{R}$ and $w_i \in \mathbb{R}^n$. We claim

$$y(x) = \sum_{i=1}^{N} v_i\sigma(w_i^T x + b_i) \tag{2}$$

may approximate any arbitrary continuous function, which is to say that

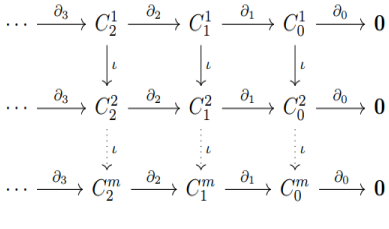
$$|y(x) - f(x)| < \epsilon \ \forall x \in I^n \Leftrightarrow y(x) \text{ are dense in } C(I^n) \tag{3}$$

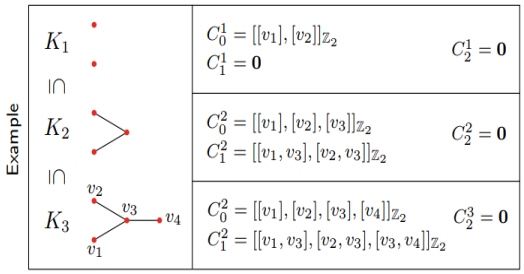
where a sigmoidal activation function $\sigma : \mathbb{R} \to \mathbb{R}$ satisfies

$$\sigma(t) = \begin{cases} 1 & t \to \infty \\ 0 & t \to -\infty \end{cases}$$

## 2 Persistent Homology

Let $(K^i)_{i=0}^m$ be a sequence of simplicial complexes such that $\emptyset = K^0 \subseteq K^1 \subseteq ... \subseteq K^m = K$. We call $(K^i)_{i=0}^m$ a *filtration* of K and $m$ the *scale*. If we set $C_n^i = C_n(K_n^i)$, where $K_n^i$ is the n-skeleton of a simplicial complex $K^i$, then we obtain the following sequence of chain complexes:



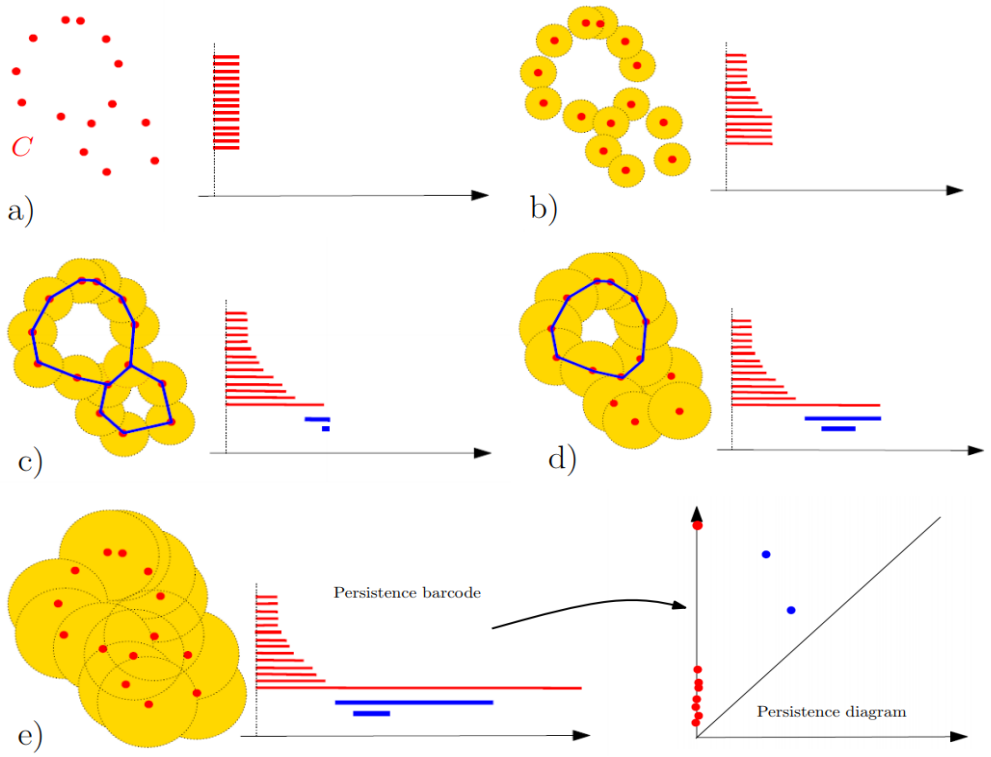




Now define persistent homology groups as

$$H_n^{i,j} = ker\ \partial_i^n \ / \ im\ \partial_j^{n+1} \cap ker\ \partial_i^n \tag{4}$$

for $i \leq j$. The ranks, $\beta_n^{i,j} = rank\ H_n^{i,j}$, of these homology groups are known as the $n^{th}$ *persistent Betti numbers*, and they capture the number of homological features of dimension $n$. For example, these ranks capture the number of connected components when $n = 0$, holes when $n = 1$, that persist from $i$ to $j$. Intuitively, as we vary the scale, new connected components can appear, different components may merge, new holes could appear or disappear, etc. These components are just the simpleces $\sigma^i \in K^i$, and if $\sigma^i$ continually reappears in several filtrations $K^j, K^{j+1}, ..., K^{j+t}$ where $j = i + 1$, we may consider $\sigma^i$ to be a significant feature of the data. Indeed, persistent homology tracks these changes as the scale varies by constructing a *persistence diagram* $D$, which is a set of tuples, where the first element in each tuple is the starting point and the second element is the ending point of a particular homological feature. For example, If the scale $r$ is large enough so that two points are within each others' radius, then a 1-simplex would be a homological feature at that scale.

**Example.** Let $C$ be a finite set of points and $r$ be the radius or scale. The following set of figures demonstrate how homological features change with $r$ and how these changes are encoded into a persistence diagram, lies on the bottom right hand side. The right hand side of each figure shows the persistence barcode, which represents the lifespan of each feature.




a) We have the radius $r = 0$, so we have only 0-simpleces, and we create a time interval for the birth of each feature at $r = 0$. As an example, if we say that $C$ is a topology, we have $H_0(C) = \mathbb{Z}^{14}$, so our $0^{th}$ Betti number is 14.

b) At some $r > 0$, the radii of these balls intersect, creating 1-simpleces that result in the death of the 0-simpleces that the 1-simpleces are composed of.

c) Now we have extended the radius so that the line segments generate two cycles, which we assign a lifespan in blue.

d) One of the two cycles has been filled, resulting in its death at the end of the corresponding blue interval.

e) All cycles have disappeared, and only a singular infinitely long red interval remains. Intuitively, the longer each interval is, the more persistent that homological feature is across the scale. So, the persistence diagram can be seen as a multi-scale topological signature encoding the homology of the union of balls for all radii as well as its evolution across the values of $r$.

## 3 Integrating persistence diagrams into a Neural Network

We introduce a network layer parametrized by learnable vectors $\mu_i$, and $\sigma^i$, called the locations and scales. Let $D = \{(x_0, x_1) \in \mathbb{R} : x_1 \geq x_0\}$ be a persistence diagram. We then let $b_0$ and $b_1$ be unit vectors in the directions $(1,1)^T$ and $(-1,1)^T$ and define a mapping $x = (x_0, x_1) \rightarrowtail (\langle x, b_0\rangle, \langle x, b_1\rangle)$. This mapping rotates points in $D$ clockwise by $\frac{\pi}{4}$. We define this mapping because it allows us to define partial derivatives with respect to our parameters, the location $\mu = (\mu_0, \mu_1)^T \in \mathbb{R}^+ x \mathbb{R}^+$ and scale $\sigma = (\sigma_0, \sigma_1)^T \in \mathbb{R} \times \mathbb{R}^+$, that are continuous and absolutely bounded. Let $v \in \mathbb{R}^+$. We define $S_{\mu,\sigma,v} : \mathbb{R} \times \mathbb{R}^+ \to \mathbb{R}$ as

$$s_{\mu,\sigma,v}((x_0, x_1)) = \begin{cases} e^{-\sigma_0^2(x_0-\mu_0)^2 - \sigma_1^2(x_1-\mu_1)^2} & x_1 \in [v, \infty) \\ e^{-\sigma_0^2(x_0-\mu_0)^2 - \sigma_1^2(ln(\frac{x_1}{v})v+v-\mu_1)} & x_1 \in (0, \infty) \\ 0 & x_1 = 0 \end{cases}$$

We project a persistence diagram $D$ via the projection $S_{\mu,\sigma,v}$ defined as

$$D \rightarrowtail \sum_{x \in D} s_{\mu,\sigma,v}(p(x)) \tag{5}$$

where $p$ is the rotation clockwise by $\frac{\pi}{4}$ that was defined in the first paragraph of the section.

Let $N \in \mathbb{N}$, $\theta = (\mu_i, \sigma_i)_{i=0}^{N-1} \in ((\mathbb{R} \times \mathbb{R}^+) \times (\mathbb{R}^+ \times \mathbb{R}^+))$. We define the concatenation of all mappings defined by the previous equation as

$$S_{\theta,v} : D \to (\mathbb{R}_0^+)^N \; D \rightarrowtail (S_{\mu_i,\sigma_i,v}(D))_{i=0}^{N-1} \tag{6}$$

Since $S_{\mu,\sigma,v}$ is a finite sum of $s_{\mu_i,\sigma_i,v}$, each of which are differentiable operations, then this network layer is compatible with the weight update rule we described in the first section of the paper. Therefore, we can train a neural network with this layer via the Backpropagation algorithm.

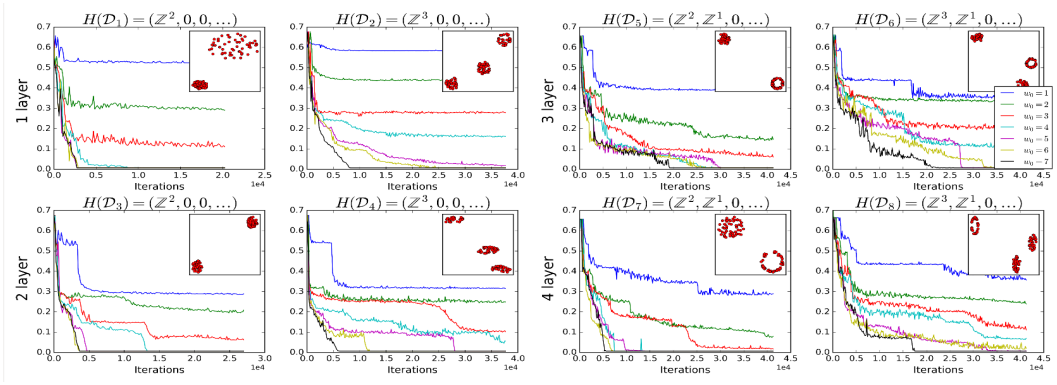
## 4 Characterizing the Expressivity of a Neural Network using Algebraic Topology

**Introduction.** It is known that the performance of a particular architecture of a Neural Network hinges on the complexity of the dataset on which it is acting. Unfortunately, describing datasets in way in which it is helpful in making the design choices of a Neural Network is a largely unsolved problem. However, Algebraic Topology provides an useful set of tools that assist in tackling this problem. Let us begin the description of these tools by first making some definitions.

**Definitions.** Let $D$ be a dataset where its points are drawn from a joint distribution with continuous CDF on a topological space $X \times \{0, 1\}$. Let $\mathbb{F} = \{f : X \to \{0, 1\}\}$ be a set of arbitrary binary classifiers on $X$. Let $X^+$ and $X^-$ denote the points in $X$ which are classified as positive and negative by some $f \in \mathbb{F}$, respectively. Let $H_S(f)$ denote the homology of the set of $x \in X$ such that $f(x) = 1$. Intuitively, $H_S(f)$ characterizes how many holes there are in the positive decision region of $f$. We will denote $\beta_n(f)$ to be the $n^{th}$ Betti number of this homology.

**Theorem 1.** If $X = X^+ \sqcup X^-$ and $\forall f \in \mathbb{F}\; H_S(f) \neq H(X^+)$, then $\forall f \in \mathbb{F}$ there exists $A \subset X^+$ such that $f$ misclassifies each $x \in A$.

Theorem 1 states that as long as the homology of the classifier at hand does not match that of the subset of the dataset which has positive classification, then there will always be exist some subset of positive examples which will all be classified as negative! Theorem 1 begs the question, can we select a neural network $N$ with architecture $A$ such that $H_S(N_A) = H(D)$, for some given dataset $D$? The answer to this question depends on some deeper problems in Algebraic Topology that are not yet resolved. Therefore we learn about how the homology of the data determines expressive architectures in an empirical manner We vary the number of layers and hidden units to measure how a network can learn and express homology on datasets of varying homological complexity.



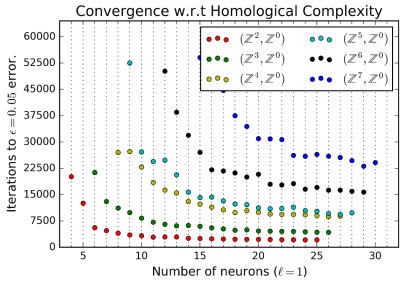
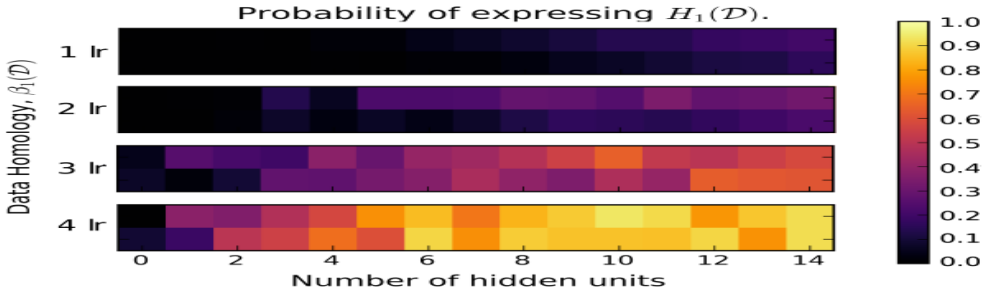




Figure 2: Another plot that supports the analysis on the right hand side, at least experimentally

In the figure above, we have the convergence rates of several 1-4 layer neural networks which vary on the homology of the dataset, where the number of neurons in the first hidden layer is increased. Above each plot, we have the $0^{th}$, $1^{st}$, ..., $n^{th}$ homologies for the data, which is pictured on the top right hand side of each plot. The left hand side of the figure denotes the error.

Notice that the more homologically complex a dataset is, the longer it takes for a network to converge, and may end up convering to a higher error rate. This analysis yields a statisticially significant observation that if the number of neurons in a network don't meet some threshhold based on the homological complexity of the data, then the neural network will not converge to a desirable error. We call this threshhold $hphase$. The approximation of $hphase$ is the motivation for the following discussion.

We apply persistent homology to find the homological features in a dataset. Let $D'$ be a persistence diagram and $(b_i, d_i) \in D'$ be the birth and death time of a homological feature. Restricting the analysis to 1-layer neural networks, we regress a multilinear model training on pairs $(b_i, d_i) \rightarrowtail argmin_m min\{\frac{B_p(f_m)}{\beta_p(D)}, 1\}$, where $\beta_p(D)$, where $\beta_p(D)$ is the $p^{th}$ Betti number of the dataset $D$, over all m hidden unit single layer neural networks $f_m$. The results of this approach are demonstrated in the plot below.




In conclusion, the paper demonstrated that the expressivity of an architecture is largely determined by the homological complexity of the data on which it acts. However, more fine-grained or reliable methods of making architectural choices based on this information do not yet exist, and is fertile ground for future research. In terms of theory, the approximation for the lower bound of $hphase$ based on the homological complexity of the data is also an open problem.

## References


[1] Russell, S. J. Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, New Jersey 07632

[2] Hofer, C., Kwitt, R., Niethammer, M., and Uhl, A. (2017) *Deep Learning with Topological Signatures.* arXiv preprint arXiv:1707.04041.

[3] Chazal, F. and Michel, B. (2017) *An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists.* arXiv preprint arXiv:1710.04019.

[4] Guss, W. Salakhutdinov, R. (2018) *On Characterizing the Capacity of Neural Networks using Algebraic Topology.* arXiv preprint arXiv:1802.04443.

[5] Goodfellow, I. Yoshua, B. Courville, A. (2016) *Deep Learning.* MIT Press.

[6] Hatcher, A. (2002) *Algebraic Topology.* Cambridge University Press.