Term Paper

Anonymous Author(s)
Affiliation
Address
email

Abstract

Deep neural networks, given enough input data, may emulate any arbitrary continuous function. In practice, they may: diagnose cancers more accurately than an oncologist, beat the world champion of the game "Go", and beat professional players in the video game "Dota 2". 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 a piece of the current research in this sub-field at the intersection of Deep Learning and Computational 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_θ, where $\theta \in \Omega$ and Ω 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* θ_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 θ_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 θ_1 is made up of the weights "$w_{1,3}$", "$w_{1,4}$", "$w_{2,3}$", "$w_{2,4}$".

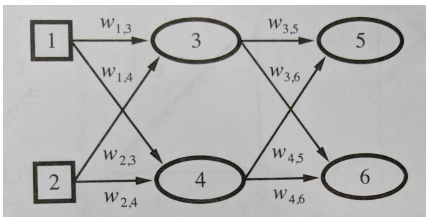


Figure 1: Feedforward Neural Network

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_θ and f^* is minimized. There's many error metrics that can be used, but for pedagogical purposes we show how f_θ 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 θ appropriately. The change of the L_2 loss with respect to θ 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 θ as follows: $\theta \leftarrow \theta - \alpha\frac{\delta}{\delta\theta}L_2$, where α is known as the *learning rate*. Intuitively, α controls how large the "jumps" we are making down the optimization landscape of θ 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 σ 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

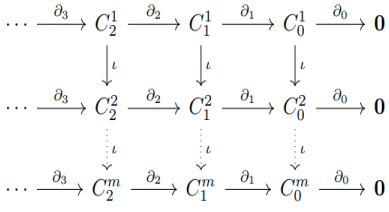
$$|y(x) - f(x)| < \epsilon \ \forall x \in I^n \Leftrightarrow y(x) \text{ are dense in } C(I^n) \tag{3}$$

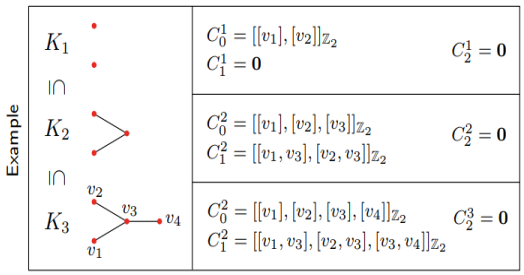
where a sigmoidal activation function $\sigma : \mathbb{R} \to \mathbb{R}$ satisfies

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

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

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