Functional Neural Networks

Approximated by Weierstrass Polynomials

Guss, William
wguss@berkeley.edu

June 11, 2015

Abstract

In this paper we consider the traditional model of feed-forward neural networks proposed in (McCulloch and Pitts, 1949), and using intuitions developed in (Neal, 1994) we propose a method generalizing discrete neural networks as follows. In the standardized case, neural mappings $\mathcal{N} : \mathbb{R}^n \to [0,1]^m$ have little meaning when $n \to \infty$. Thus we consider a new construction $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$ where the domain and codomain of $\mathcal{N}$ become infinite dimensional Hilbert spaces, namely the set of quadratically Lebesgue integrable functions L^2 over a real interval E and $[0,1]$ respectively. The derivation of this construction is intuitively similar to that of Lebesgue integration; that is, $\sum_i \sigma_i w_{ij} \to \int_{E \subset \mathbb{R}} \sigma(i) w(i,j)\, d\mu(s)$.

After establishing a proper family of "functional neural networks" $\mathcal{F}$, we show that $\mathcal{N}$ are a specific class of functional neural networks under specific constraints. More specifically in our first lemma, we prove that $\mathcal{F} \equiv \mathcal{N}$ for piecewise constant weight functions $w(i,j)$. Having done so, we then attempt to find an analogue to Cybenko's theorem of universal approximation for neural networks. Firstly, we prove as a corollary of the Weierstrass approximation theorem, that $w(i,j)$ can approximate a function, $f : E \to [0,1]$, satisfying $\|\mathcal{F}\xi - f(\xi)\|_\infty \to 0$. As a byproduct of the proof, we also establish a closed-form definition for the satisfying $w(i,j)$ and thereby through our first lemma provide novel insight into the actual form of the weight matrix $[w_{ij}]$ for trained $\mathcal{N}$. Finally we propose a universal approximation theorem for functional neural networks; that is, we show through the Riesz Representation Theorem that $\mathcal{F}$ approximates any bounded linear operator on $\mathcal{X}$.

In conclusion, we create a practical analogue of the error-backpropagation algorithm, and implement functional neural networks using Simpsons rule. We suggest that functional neural networks represent an interesting opportunity for the implementation of machine learning systems modeling functional transformation.

Contents

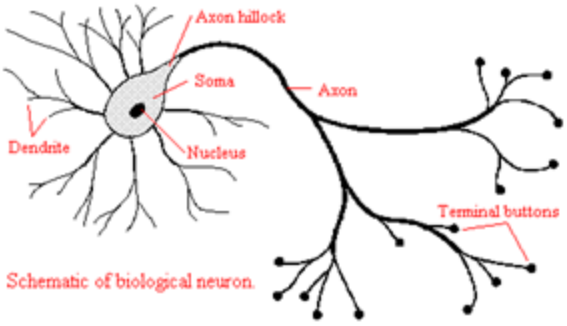

Figure 1: A biological neuron **neuralimage**

1 Introduction

Machine learning is an emerging field that deals with the development of algorithms which can predict and classify novelties based on a set of prior intuitions **mlsurvey** The field incorporates ideas from biology, computer science, numerical analysis, and statistics. In recent years machine learning has entered the main stream through web services like Google, Facebook, and Amazon. There is an incentive from both academia and industry to expand machine learning techniques and applications.

One of the most popular algorithms of the field is the artificial neural network (ANN). Although there are numerous mathematical interpretations of neural networks, we will primarily focus on the expansion of one type, feed-forward neural networks. As ANNs are based off the structure of biological neurons, a biological approach is necessary to understand this interpretation.

1.1 Biological Neuron

A single neuron consists of the cell body (the soma), the dendrites, and the axon. Mathematically we wish to examine the process of neural activation, the events which lead to the excitation of the axon. Consider a neuron that has activated anterior neurons (those which are connected dendritically); that is, the neuron is receiving input along all of its dendrites. These electrical inputs propagate through the dendrites and become integrated on the soma as electrical membrane potential **griffith** The soma then acts as the primary computational unit and activates the axon when a threshold of input activity is reached. More specifically, when a membrane potential of about -60 mV is reached on the soma, the hillock zone, or axon hillock, activates the axon by applying proteins to an ion channel which creates action potential along the axon **bioneuron**

1.2 Artificial Neurons

With this in mind it is now possible to construct a mathematical model of an artificial neuron. Let A_j, P_j be the set of anterior and posterior neurons of a neuron, j. Then, the cell membrane naturally becomes a linear combination of the dendritic potentials.

Definition 1.2.1. *We say that* net_j *is the net electric potential over the membrane, if for*

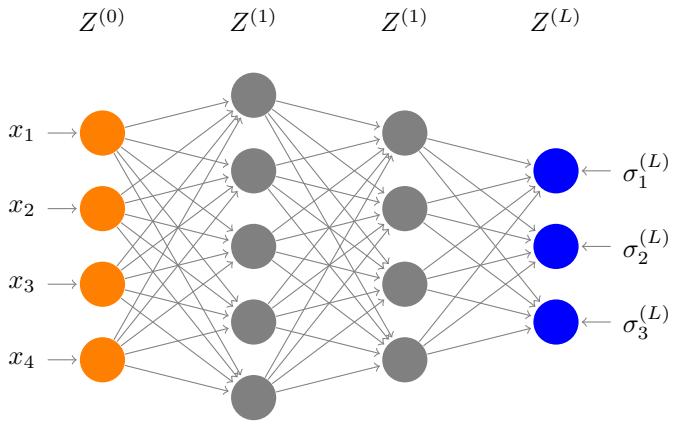

Figure 2: An example of a feed-forward ANN, $\mathcal{N}$ with four layers.

a natural neural resting potential β,

$$net_j = \sum_{i \in A_j} w_{ij}\sigma_i + \beta$$

where w_{ij} *is the dendritic connection strength from the* i^{th} *anterior neuron to* j, σ_i *is the action potential being propagated from the* i^{th} *anterior neuron.*

Furthermore, the thresholding of the hillock zone is given by some real valued sigmoidal function g bijective and differentiable over $\mathbb{R}$.

Definition 1.2.2. *We call* σ_j *the action potential of a neuron* j *if*

$$\sigma_j = g\left(\text{net}_j\right)$$

for some continuous real valued, monotonically increasing function g.

This model of the artificial neuron follows from the work that Pitt and McCulloch did in representing neural activity as logical thresholding elements **mcculloch**

1.3 Feed-Forward Artificial Neural Networks

Now we have a sufficient mathematics base to define the feed-forward artificial neural network. The concept of a feed-forward ANN is biologically motivated by the functional organization of the visual cortex. It is appropriate to divide the structure of the visual cortex into layers which are denoted V1, V2, V3, and so on. The layers are organized such that a given layer is directly adjacent to and exhibiting full connectedness to the subsequent layer, an example being V1 to V2, V2 to V3, and subsequently for all of the primary layers of the visual cortex. From a functional point of view these layers store levels of visual abstraction like lines and shapes on the lower layers to faces and abstract visual concepts on the highest layers**visualcortex**

The goal is to model this representation of increasing abstraction whilst maintain adjacency and full topological connectedness. Thus we construct a set of neural layers with cardinality $L+1$, and connections as depicted in Figure 2.

Definition 1.3.1. *We say $\mathcal{N}$ is a feed-forward neural network if for an input vector $\boldsymbol{x}$,*

$$\begin{aligned} \mathcal{N} : \sigma_j^{(l+1)} &= g\left(\sum_{i \in Z^{(l)}} w_{ij}^{(l)} \sigma_i^{(l)} + \beta^{(l)}\right) \\ \sigma_j^{(1)} &= g\left(\sum_{i \in Z^{(0)}} w_{ij}^{(0)} x_i + \beta^{(0)}\right) \end{aligned}$$

Where $1 \leq l \leq L-1$.

For mathematical convenience let us denote $\sigma_j^{(l)}$ as the output of the j^{th} neuron on layer l. In this construction we prefer three different types of neurons, the input neuron, the hidden neuron, and the output neuron. In the case of the input neuron, there is no sigmoidal activation function, and instead we assign each $\sigma_j^{(0)}$ to a real value which is then weighted by the dendritic input strength of each anterior neuron. Moreover an input neuron only exists on the 0^{th} layer. In the case of each hidden layer we adopt the model described for the standard neuron as aforementioned where our sigmoid activation function $g = \tanh(\text{net})$ is the hyperbolic tangent.Finally, the output layer usually have a linear sigmoid activation as to achieve output scaling beyond $[1, -1]$ in the previous layers. Once again the output layer can only exist on the layer L.

1.4 Error Backpropagation

With the functional organization of the network complete, we now need to develop the notion of learning. For the purposes of this paper we will describe a gradient descent method for learning called error-backpropagation. In the mathematical model we find conveniently that the degrees of freedom are then the dendritic weights between any two neurons. Thus these weights must be optimized against some desired output. This leads to the following multi-dimensional error function.

Definition 1.4.1. *We call E the error function of a neural network $\mathcal{N}$, if for an input vector $\boldsymbol{x}$*

$$E\left(w_{00}^{(0)}, w_{01}^{(0)}, \ldots, w_{ij}^{(L)}\right) = \frac{1}{2} \sum_{i \in Z^{(L)}} \left(\sigma_i^{(L)} - \delta_i\right)^2$$

where $\boldsymbol{\delta}$ is some desired output vector corresponding to $\boldsymbol{x}$.

Then the goal is to optimize this error function such that a reasonable local minimum is found. We then choose to modify each weight in the direction of greatest decrease for the error function.

Definition 1.4.2. *We call $\boldsymbol{\nabla} E$ the gradient of E if*

$$\boldsymbol{\nabla} E = \left(\frac{\partial E}{\partial w_{00}^{(0)}}, \frac{\partial E}{\partial w_{01}^{(0)}}, \ldots, \frac{\partial E}{\partial w_{ij}^{(L)}}\right)$$

for all weights, $w_{ij}^{(l)}$, in feed-forward ANN $\mathcal{N}$.

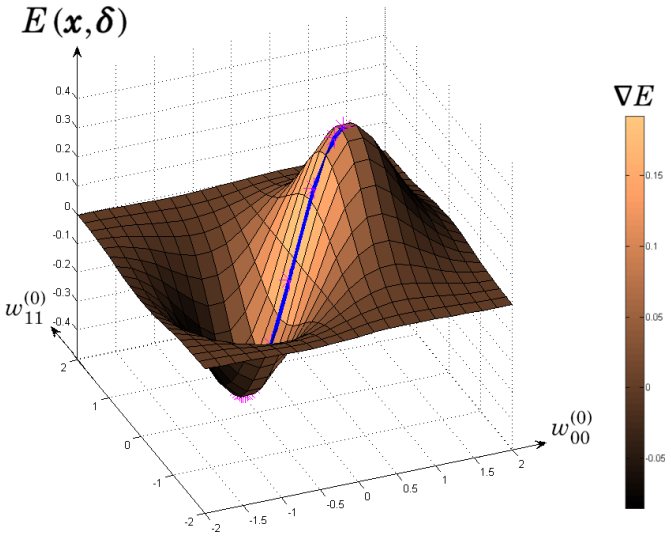

Figure 3: Gradient descent on $E(\boldsymbol{x},\boldsymbol{\delta})$ with descent path shown in blue.

Conveniently the gradient of a function describes a vector whose direction is the greatest increase of a function. Thus to optimize our weights so that the lowest error is achieved, we update the weights as follows: $\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \alpha\boldsymbol{\nabla}E$ where alpha is some learning rate, a process which is depicted in Figure 3 **rumelhart1988learning**

The calculation of each $\frac{\partial E}{\partial w_{ij}^{(L)}}$ is non-trivial given that each weight influences the error function in multiple ways. To find the contribution of a single weight, recall that every single neuron is connected to all neurons in the anterior and posterior layers. So, a single weight will influence not only its posterior neurons sigmoidal output but also that of every neuron for any path from the posterior neuron to the set of output neurons. Thus, the multiplicity of contribution via different neural routes follows directly from the multidimensional chain-

rule. Recall the differential operator $D_g f = \frac{\partial f}{\partial x}$,

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}^{(l)}} &= D_{\sigma_0^{(L)}} \cdot D_{\text{net}}\sigma_0^{(L)} \cdot D_{\sigma_0^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&+ \ D_{\sigma_1^{(L)}} \cdot D_{\text{net}}\sigma_1^{(L)} \cdot D_{\sigma_0^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&\ \ \vdots \\
&+ \ D_{\sigma_n^{(L)}} \cdot D_{\text{net}}\sigma_n^{(L)} \cdot D_{\sigma_0^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&+ \ D_{\sigma_0^{(L)}} \cdot D_{\text{net}}\sigma_0^{(L)} \cdot D_{\sigma_1^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&\ \ \vdots \\
&+ \ D_{\sigma_n^{(L)}} \cdot D_{\text{net}}\sigma_n^{(L)} \cdot D_{\sigma_1^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&\ \ \vdots \\
&+ \ D_{\sigma_n^{(L)}} \cdot D_{\text{net}}\sigma_n^{(L)} \cdot D_{\sigma_m^{(L-1)}}\text{net} \cdot \ \cdots \ \cdot D_{\text{net}}\sigma_j^{(l+1)} \cdot D_{w_{ij}^{(l)}}\text{net} \\
&= \sum_{a_1}^{Z^{(L)}} \sum_{a_2}^{Z^{(L-1)}} \cdots \sum_{a_m}^{Z^{(l+2)}} \frac{\partial E}{\partial \sigma_{a_1}^{(L)}} \frac{\partial \sigma_{a_1}^{(L)}}{\partial \text{net}} \frac{\partial \text{net}}{\partial \sigma_{a_2}^{(L-1)}} \ \cdots \ \frac{\partial \sigma_{a_m}^{(l+2)}}{\partial \text{net}} \frac{\partial \text{net}}{\partial \sigma_j^{(l+1)}} \frac{\partial \sigma_j^{(l+1)}}{\partial \text{net}} \frac{\partial \text{net}}{\partial w_{ij}^{(l)}}
\end{aligned} \tag{1.4.1}$$

At first sight this algorithm looks quite complicated, but a computational implementation would be able to cache certain sums and essentially reduce the complexity thereof. With the error backpropagation algorithm complete, the notion of an artificial neural network, its training, and processing is complete. Now it is possible to conjecture on variants thereof and present the primary scope of this essay.

1.5 The Research Question

This construction is of serious mathematical interest as feed-forward neural networks have been shown to be universal approximators; that is, they can approximate any $f : A \to B$, where $A, B \in \mathbb{R}^m$ are vector spaces. However it is not certain what information the approximation provides: the gradient descent algorithm does not reveal any connections between the values of the inputs. A standing question in the field asks what can be said about the weights satisfying an approximation of f besides that they exist. Therefore it is of considerable interest to explore the general form of the weights as training may not be necessary and computational complexity can be lowered.

It is the subject of this research essay to explore the neural networks through a mathematical exploration. To do this, a technique from economic mathematics is employed. It is typical that to analyze a discrete economic model, time is considered continuous and summations become integrals. To investigate neural networks then, this technique will be applied. Thus the question arises: **what can be said about artificial neural networks as the number of nodes approaches infinity and how can a real valued, continuous analogue for neural networks contribute to or aid in understanding the black box model of artificial neural networks?**

In this paper we will generalize the notion of the universal approximation for arbitrary vector space mappings to arbitrary approximation of any $f : L^1(\mathbb{R}^n) \to C^\infty(\mathbb{R}^n)$ by examining the structure of feed-forward ANNs as the number of nodes for each layer becomes uncountably bounded in $\mathbb{R}^n$. Such a generalization requires that a continuum of neural components be made, and that a continuous weight tensor or hypersurface must exist in order to maintain the topological connectedness as prescribed by the discrete model.

2 Functional Neural Networks

Although neural networks have proven an extremely effective mechanism of machine learning **mlsurvey** theoretically they remain a black-box model. In answer to this problem Neal examined the notion of infinite hidden nodes with a network proving that such a construction becomes a Gaussian kernel. Then, Roux and Bengio described a model for affine neural networks with continuous hidden layers in alignment with Neal's research. These authors showed effectively the viability of a "continuous" neural network, but left many similar constructions unexplored. It is the subject of this paper to generalize the construction of a feed-forward ANN which maps uncountably infinite vector spaces, and then to demonstrate the practical implementations of algorithms such as error backpropagation in this generalized form.

2.1 The Core Idea

Suppose that we wish to map one functional space to another with a neural network. Consider the standard model of an ANN as the number of neural nodes for every layer becomes uncountable. The index for each node then becomes real-valued, along with the weight and input vectors . Let us denote $\xi : \mathbb{R}^n \to \mathbb{R}$ as some arbitrary input function for the neural network. Likewise consider a real-valued weight function, $w^{(l)} : \mathbb{R}^{2n} \to \mathbb{R}$, for a later l which is composed of two indexing variables $i, j \in \mathbb{R}^n$. Finally as the number of neural nodes becomes uncountable we define a real-valued bound for any given layer $R^{(l)} \supset Z^{(l)}$. As the indices become real valued, examination of the sigmoidal sum seen in definition 1.1 leads us to the following derivation.

$$\begin{aligned} \sigma^{(1)}(j) &= \lim_{\Delta i \to 0} g\left(\sum_{i \in R^{(0)}} \xi(i) w^{(0)}(i,j)\ \Delta i + \beta \right) \\ &= g\left(\int_{R^{(0)}} \xi(i) w^{(0)}(i,j)\ di + \beta \right) \end{aligned} \tag{2.1.1}$$

Repeating the process inductively for all layers in a neural network, leads to a recurrance relation for this construction.

Definition 2.1.1. *We call $\mathcal{F}$ a functional neural network if,*

$$\begin{aligned} \mathcal{F} : \sigma^{(l+1)}(j) &= g\left(\int_{R^{(l)}} \sigma^{(l)}(i) w^{(l)}(i,j)\ di + \beta \right) \\ \sigma^{(0)}(j) \quad &= \xi(j) \end{aligned} \tag{2.1.2}$$

To demonstrate the accuracy of this generalization, let us propose the following lemma which is near that of Roux and Bengio.

Theorem 2.1.2. *Suppose $\mathcal{F}$ is a functional neural network with a set of piecewise constant weight functions $W = \left\{ w^{(0)}, w^{(1)}, \ldots, w^{(L-1)} \right\}$ each with constituent pieces of length one. Then given the input function $\xi(i) = x_n, n = \max\{m \in \mathbb{Z} \mid m \leq i\}$ for some input vector $\boldsymbol{x}$, $\mathcal{F}$ is discretized; that is assuming $i, j \in \mathbb{R}$ and $Z^{(l)} = R^{(l)} \cap \mathbb{Z}$ then for $0 \leq l \leq L-1$ we have that*

$$\mathcal{F}(\xi) \equiv \mathcal{N}(\boldsymbol{x}) \tag{2.1.3}$$

Proof. Let $P(l)$ be the proposition that $\sigma^{(l+1)}(j)$ becomes discretized when $w^{(l)}(i,j)$ and $\sigma^{(l)}(i,j)$ are piecewise constant with constituent functions of length one. Moreover let $w_{ij}^{(l)}$ denote the value of $w^{(l)}(i,j)$ for $(i,j) = \max\left\{(x,y) \in \mathbb{Z}^2 \mid x \leq i \wedge y \leq j\right\}$. Then by induction we show $P(l), 0 \leq l \leq L-1$.

Basis Step. Recall that $\sigma^{(0)}(j) = \xi(j)$. Then one would suppose

$$\sigma^{(1)}(j) = g\left(\int_{R^{(0)}} \xi(i) w^{(0)}(i,j)\ di + \beta\right) \tag{2.1.4}$$

but because the weight function and the input function are piecewise constant and not guaranteed to be continuous for $R^{(0)}$, we must take the improper integral along each constituent piece of length one. Supposing that each summation in the following is taken over $k \in Z^{(0)}$,

$$\begin{aligned}
\sigma^{(1)}(j) &= g\left(\sum_k \lim_{b\to k} \int_{k-1}^{b} \xi(i) w^{(0)}(i,j)\ di + \beta\right) \\
&= g\left(\sum_k w_{kj}^{(0)} \lim_{b\to k} \int_{k-1}^{b} \xi(i)\ di + \beta\right) \\
&= g\left(\sum_k w_{kj}^{(0)} x_b + \beta\right)
\end{aligned} \tag{2.1.5}$$

Inductive Step. Now assume that for some l we have that

$$\sigma^{(l+1)}(j) = g\left(\sum_{i \in Z^{(l)}} w_{ij}^{(l)} \sigma_i^{(l)} + \beta\right) \tag{2.1.6}$$

We now show that by the inductive hypothesis if $P(0) \wedge P(l) \to P(l+1)$, then $P(k)\ \forall k$. Consider the next neural layer defined as

$$\sigma^{(l+2)}(j) = g\left(\int_{R^{(l+1)}} \sigma^{(l+1)}(i) w(i,j)\ di + \beta\right) \tag{2.1.7}$$

Then because $w(i,j)$ and $\sigma^{(l+1)}$ are piecewise constant by definition and not necessarily continuous for $R^{(l+1)}$ we must again take the improper Riemann integral over the constituent

pieces. Consider $k \in Z^{(l+1)}$

$$\begin{aligned}\sigma^{(l+2)}(j) &= g\left(\sum_k \lim_{b\to k}\int_{k-1}^{b} \sigma^{(l+1)}(i)w(i,j)\ di + \beta\right) \\ &= g\left(\sum_k w_{kj}^{(l+1)}\sigma_k^{(l+1)} \lim_{b\to k}\int_{k-1}^{b} di + \beta\right) \\ &= g\left(\sum_k w_{kj}^{(l+1)}\sigma_k^{(l+1)} + \beta\right)\end{aligned} \tag{2.1.8}$$

Therefore by the inductive hypothesis the proof follows and $\mathcal{F}(\xi) \equiv \mathcal{N}(\boldsymbol{x})$ for piecewise constant input and weight functions.

□

From the logic of the preceding proof we can establish that, the input function need only be properly Lebesgue integrable over $R^{(0)}$. Moreover, we come to an extremely important intuition; the weight matrix for a given layer l can be thought of as a piecewise constant weight surface, and the linear combination of weights can be thought of as a piecewise integral transformation along a given j on the weight surface. However, functional neural networks allow for infinite weight surfaces and therefore can represent the entire class of integral transforms. With this in mind, it is now possible to consider functional neural networks as universal approximators.

2.2 Universal Approximation of Bounded Linear Operators

In the case of discretized neural networks, George Cybenko and Kolmogorov have shown that with sufficient weights and connections, a feed-forward neural network is a universal approximator of arbitrary $f : \mathbb{R}^n \to \mathbb{R}^m$ **univapprox** that is, constructs of the form $\mathcal{N}(\boldsymbol{x})$ are dense in $C(I^n, \mathbb{R}^{\geq})$ where I^n is the unit hypercube $[0, 1]^n$. Cybenko proved this remarkable result by utilizing the Riesz Representation Theorem for Hilbert spaces and the Hahn-Banach theorem. He showed by contradiction that there exists no bounded linear functional $h(x)$ in the form of $\mathcal{N}(\boldsymbol{x})$ such that $\int_{I_n} h(x)\ d\mu(x) = 0$.

Although this theorem has led to a remarkable interest in ANNs by legitimizing their effectiveness, it still has shed no light on the internal workings of ANNs. Fortunately using the intuitions presented in Theorem 1, it would be advantageous to examine the generalization of Cybenko's theorem to the larger class of functional neural networks. However, there is no clear way with which to do this; discretized neural networks map vector spaces and therefore approximate continuous functions, whereas functional neural networks are defined as arbitrary mappings between Hilbert spaces (more specifically the set of L^2 integrable functions). Moreover letting n approach infinity in Cybenko's proof fails to hold in that there is not an obvious topology for $C(C(\mathbb{R}), C(\mathbb{R}))$. Therefore we must develop an approximation theorem for $\mathcal{F} : C(X) \to C(Y)$ over the set of linear operators.

First, however, let us develop the notion of $\mathcal{F}$ as a universal approximator of arbitrary functions. By Theorem 1, we have that $\mathcal{F} \equiv \mathcal{N}$,and in that sense for piecewise constant $w(i, j)$ and $\xi(i)$ functional neural networks approximate any arbitrary mapping from $\mathbb{R}^n \to \mathbb{R}$

where $n = |Z^{(0)}|, m = |Z^{(L-1)}|$ by Cybenko's theorem. However, when considering the fully continuous case the following corollary arises from the Stone-Weierstrass theorem.

Corollary 2.2.1. *Suppose $\mathcal{F}$ is a multi-layer functional ANN. Then for some real-valued continuous function $f : \mathbb{R} \to \mathbb{R}$, there exists a set of weights W such that $\forall \epsilon > 0$, $\|\mathcal{F}(\xi) - f\|_\infty < \epsilon$*

Proof. In this proof we will ommit the inductive step as it is elementary and employs the same logic as the basis step. Consider the first neural layer

$$\sigma^{(1)}(j) = g\left(\int_{R^{(0)}} \xi(i) w^{(0)}(i,j)\ di\right) \tag{2.2.1}$$

because we take $w^{(0)}$ to be some approximating polynomial by the Stone-Weierstrass theorem, let $w^{(0)} = \left[(g^{-1})' \circ (h(\Xi, j))\right] h'(\Xi, j)$ approximately, where Ξ is the primitive of ξ. Supposing that h is some polynomial satisfying $h(\Xi, j)\Big|_{R^{(0)}} = f(j)$, then by the chain rule of integration

$$\begin{aligned} \sigma^{(1)}(j) &= g\left(\int_{R^{(0)}} \xi(i) \left[(g^{-1})' \circ (h(\Xi, j))\right] h'(\Xi, j)\ di\right) \\ &= g\left(g^{-1}\left(h(\Xi, j)\right)\right)\Big|_{R^{(0)}} \\ &= f(j) \end{aligned} \tag{2.2.2}$$

□

At this point it is important to note that the proof given above implies that ξ is disregarded through manipulation of $w^{(0)}$. Instead, h is a function of Ξ which is the primitive of ξ. If we were to not let $h(\Xi, j)\Big|_{R^{(0)}} = f(j)$, then we could claim that for arbitrary h we have proven any functional composition of ξ is possible; that is, in some light sense we have proven Cybenko's theorem in the general case by treating the weight set as some hypersurface. Moreover, the intuition follows that if we were to discretize the satisfying h and ξ (Theorem 1) then it is possible that a similar weight surface is developed for a trained $\mathcal{N}$. This result is remarkable as new light is shed on the black-box model of neural networks showing that approximation of h is made in the discrete sense.

Although we have shown through the corollary that approximation of arbitrary functional composition is possible, we have yet to consider values of w in the general sense. In other words, what can be said about the general approximation of bounded linear operators mapping $C(\mathbb{R}^n)$ to $C(\mathbb{R}^n)$ where C denotes the set of continuous (integrable) real valued functions? Evidently, the form of $\mathcal{F}$ resembles the general class of integral transforms, $\int f(x) \cdot E(x, k)\ dx$. Integral transforms are shown to approximate a multitude of operators through varying kernel functions. For example consider some $g(t)$ and the integral transform

$$(\mathcal{P}g)(s) = \int_0^\infty g(t)\delta'(s - t)\ dt \tag{2.2.3}$$

where δ is the Dirac-Delta function. Then we have that $(\mathcal{P}g)(s) = \frac{dg}{dt}\Big|_s$ by the properties of the Delta function. Similar approximations of linear operators can be made by varying the

kernel function E. Thus there is considerable interest in determining the density of integral transforms and thereby functional neural networks in the set of bounded linear operators.

Further investigation into dense forms of bounded linear operators leads us to the Schwartz theorem of bounded linear operator representation by integral kernels. The theorem simply states that all linear operators can be represented in some light sense by integral transforms with arbitrary kernels. However, this theorem is too general for our purposes and we would like to show that in the specific case of some functional neural network $\mathcal{F}$ that for any given layer such that $l \neq 0$ any linear operator can be approximated with point-wise convergence from the Weierstrass polynomial approximation theory.

In order to do this we will return to the Riesz representation theorem that states the following **revisited**

Theorem 2.2.2. *Let $\phi : C(X) \to \mathbb{R}$ be any bounded linear form where X is a compact Hausdorff space and $C(X)$ is the Banach space of continuous functions over X. Then there exists a unique regular Borel measure μ on X such that*

$$\phi(f) = \int_X f(t)\ d\mu(t), \quad f \in C(X), \quad t \in X \tag{2.2.4}$$

and $\|\phi\| = |\mu|(X)$ where $|\mu|$ is the total variation of μ on X.

As opposed to generalizing Cybenko's theorem to Banach spaces ($\mathbb{R}^\infty$), we can actually manipulate the representation theorem to encapsulate bounded linear operators over locally compact Hausdorf spaces. Using the aforementioned logic the universal representation theorem for functional neural networks is now proposed.

Theorem 2.2.3. *Given a functional neural network $\mathcal{F}$ then some layer $l \in \mathcal{F}$, the let $K : C(R^{(l)}) \to C(R^{(l)})$ be a bounded linear operator. If we denote the operation of layer l on layer $l-1$ as $\sigma^{(l+1)} = g\left(\Sigma_{l+1}\sigma^{(l)}\right)$, then for every $\epsilon > 0$, there exists a weight polynomial $w^{(l)}(i,j)$ such that the supremum norm over $R^{(l)}$*

$$\left\| K\sigma^{(l)} - \Sigma_{l+1}\sigma^{(l)} \right\|_\infty < \epsilon \tag{2.2.5}$$

Proof. Let $\zeta_t : C(R^{(l)}) \to R^{(l)}$ be a linear form which evaluates its arguments at $t \in R^{(l)}$; that is, $\zeta_t(f) = f(t)$. Then because $\zeta_t$is bounded on its domain, $\zeta_t \circ K = K^\star\zeta_t$ is a bounded linear functional. Then from the Riesz Representation Theorem (Theorem 1) we have that there is a unique regular borel measure μ_t on $R^{(l)}$ such that

$$\begin{gathered}\left(K\sigma^{(l)}\right)(t) = K^\star\zeta_t\left(\sigma^{(l)}\right) = \int_{R^{(l)}} \sigma^{(l)}(s)\ d\mu_t(s), \\ \|\mu_t\| = \|K^\star\zeta_t\|\end{gathered} \tag{2.2.6}$$

Then if there exists a regular borel measure μ such that μ_t is significantly smaller that μ for all t, then we have that $d\mu_t(s) = K_t(s)d\mu(s)$ under the assumption that K_t is L^1 integrable over $R^{(l)}$ with the measure μ. Thus it follows that

$$K\left[\sigma^{(l)}\right](t) = \int_{R^{(l)}} \sigma^{(l)}(s)K_t(s)\ d\mu(s) = \int_{R^{(l)}} \sigma^{(l)}(s)K(t,s)\ d\mu(s). \tag{2.2.7}$$

Therefore, for any bounded linear operator $K : C(X) \to C(X)$ there exists a unique $K(t,s)$ such that $K[f] = \int_X f(s)K(t,s)d\mu(s)$. Now we show that the operation of Σ_l can approximate any such operator. Because K is of the form of Σ_l where the only difference is the weighting function, we assert the following claim.

Let G be defined as a linear functional applied to a gaussian heat kernel whose application with a function $f : \mathbb{R} \to \mathbb{R}$ yields the following definition,

$$G[f](x) = \frac{1}{b\sqrt{\pi}} \int_{\mathbb{R}} f(u) \mathrm{e}^{\left(\frac{u-x}{b}\right)} \, du. \tag{2.2.8}$$

Then it follows that by the Weierstrass approximation theorem that for all $\epsilon > 0$, the supremum norm $\|f - G[f]\|_\infty < \epsilon$. Then because G is a polynomial, f must be a limit of polynomials. So now consider the operation of $K\left[\sigma^{(l)}\right](t)$ with kernel $K(t, s)$. By the weierstrass approximation theorem $K(t, s)$ must be a limit of polynomials and therefore we let $w^{(l)}(i, j)$ assume that limit. That is,

$$\begin{aligned} \lim_{b\to 0} \left\| K\left[\sigma^{(l)}\right] - \int_{R^{(l)}} \sigma^{(l)}(s) \; G[k] \; d\mu(s) \right\|_\infty = \\ \left\| K\left[\sigma^{(l)}\right] - \Sigma_{l+1}\left[\sigma^{(l)}\right] \right\|_\infty < \epsilon \end{aligned} \tag{2.2.9}$$

Thus we have that as a limit of polynomials the operation of any arbitrary layer of a functional neural network $\mathcal{F}$ approaches any arbitrary linear bounded operator $K : C(R^{(l)}) \to C(R^{(l)})$; that is, functionals of the form Σ_{l+1} are dense in the set of all bounded continuous linear operators. □

In the above proof, a remarkable fact has been shown: functional neural networks can perform most mathematical operations on their inputs. Consider an actual application, signal processing. If one were to employ a functional neural network for the purpose of signal classification (ie, converting audio waveforms of actual words to some digital representation like a string), the mathematical construct could perform a variety of operations such as the Fourier transform or the discrete wavelet transform, all of which are important to signal processing. More importantly, however, the approximating nature of functional neural networks is extremely valuable because it can approximate kernels $(w(i, j))$ that perform operations undiscovered by conventional mathematical methods. This advantage is similar to that of neural networks, but applicable to both discrete and continuous datasets.

Naturally, one is concerned with how those weight functions and thereby operations are developed without the preselection of priors or specific coefficients for the weight polynomials. This concern leads to the development of a training method for functional neural networks.

2.3 Continuous Error Backpropagation

As described for discretized feed forward neural networks, a commonly used training method is the error-backpropagation algorithm. Therefore, it would be ideal if such an algorithm could be developed for arbitrary $\mathcal{F}$. The traditional algorithm chiefly employs a complicated expansion of the multivariate chain rule, in the case of functional neural networks, such an expansion would not be necessary because the composition from layer to layer is continuous not discretely summed. For $\mathcal{F}$ the same training can be achieved through a likewise derivation.

First, however, as were for discretized neural networks, the parameters of functional neural networks are defined. In particular, the weight surface for each layer is ideally optimized for a desired output operator. To construct these weight surfaces we will parameterize them using coefficients of Weierstrass polynomials. Consider the following definition.

Definition 2.3.1. *We say that a weight function is parameterized if it is defined as follows.*

$$w^{(l)}(i,j) = \sum_{x_{2l+1}}^{Z_Y^{(l)}} \sum_{x_{2l}}^{Z_X^{(l)}} k^{(l)}_{x_{2l},x_{2l+1}} j_l^{x_{2l}} j_{l+1}^{x_{2l+1}} \tag{2.3.1}$$

where $k_{x_{2l},x_{2l+1}}$ is some real valued coefficient and the order of the polynomial is arbitrary.

Using the above definition let us construct and error function for a functional neural network.

Definition 2.3.2. *For a functional neural network $\mathcal{F}$ and a dataset $\{(\gamma_n(j), \delta_n(j))\}$ we say that the error for a given n is defined by*

$$E = \frac{1}{2} \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right)^2 \, dj_L \tag{2.3.2}$$

This error function is seemingly close to that given in the discrete case, only a continuation and integral have been supplemented to appropriate the algorithm for functional neural networks. Now with that in mind it follows from .5 that a proper gradient descent algorithm is given. It is now essential to calculate $\frac{\partial E}{\partial k^{(l)}_{x,y}}$ in order to optimize the coefficient for our desired output function δ_n. The calculation is given as follows

$$\begin{aligned}
\frac{\partial E}{\partial k^{(l)}_{x,y}} &= \frac{\partial}{\partial k^{(l)}_{x,y}} \frac{1}{2} \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right)^2 \, dj_L \\
&= \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right) \frac{\partial}{\partial k^{(l)}_{x,y}} \mathcal{F}(\gamma_n) \, dj_L \\
&= \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right) g' \left(\int_{R^{(L-1)}} \sigma^{(L-1)}(j_{L-1}) w^{(L-1)}(j_{L-1}, j_L) \, dj_{L-1} \right) \frac{\partial}{\partial k^{(l)}_{x,y}} \cdots \, dj_L \\
&= \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right) g' \left(\Sigma_L \sigma^{(L-1)} \right) \int_{R^{(L-1)}} w^{(L-1)}(j_{L-1}, j_L) \frac{\partial}{\partial k^{(l)}_{x,y}} \sigma^{(L-1)} \, dj_{L-1} \, dj_L \\
&= \int_{R^{(L)}} \left(\mathcal{F}(\gamma_n) - \delta_n\right) g' \left(\Sigma_L \sigma^{(L-1)} \right) \int_{R^{(L-1)}} w^{(L-1)} g' \left(\Sigma_{L-1} \sigma^{(L-2)} \right) \int_{R^{(L-2)}} \\
&\quad \cdots \int_{R^{(l)}} \sigma i^x j^y dj_l \cdots dj_L
\end{aligned} \tag{2.3.3}$$

The above definition of the partial derivative of the error function can be used for the typical gradient descent method described earlier in the paper. With the continuous error backpropagation algorithm complete, the construction of functional neural networks is finished.

3 Fast Functional Neural Networks

Now that the functional neural network has been theoretically defined, the next step is to determine whether the implementation of a numerical algorithm is possible. The problem is approached using techniques already developed for discretized networks. To produce an output for a network with L layers, the implementation would propagate through all the layers resulting in $O(n^L)$ complexity. Because this is the complexity after only one propagation through the network, multiple trials will result in an immense computational time requirement. Through the definition of numerical integrability, the complexity can be easily reduced.

Definition 3.0.3. *Let $f : \mathbb{R}^{2n} \to \mathbb{R}$, We say that some function ϕ of the form*

$$\phi(x) = \int_{E\subset\mathbb{R}^n} f(x,y)dy \tag{3.0.4}$$

can be numerical integrated if and only if

$$\phi(x) = h(x)\int_{E\subset\mathbb{R}^n} /g(y)dy \tag{3.0.5}$$

for some $h, g : \mathbb{R}^n \to \mathbb{R}$.

It is clear that this definition will be useful in the establishment of an algorithm for the feed forward action of functional neural networks. It is essential for numerical quadrature to remove all free variables for calculation.

3.1 Fast Forward Propagation

In this section, the integrability of the feed forward network is shown through the following theorem. By showing that the feed forward action of functional neural networks is numerically integrable we can then develop a practical algorithm and perform a more rigorous computational complexity analysis.

Theorem 3.1.1. *If $\mathcal{F}$ is a functional neural network with L consecutive layers, then given any l such that $0 \le l < L$, $\sigma^{(l+1)}$ is numerically integrable, and if ξ is any continuous and Riemann integrable input function, then $\mathcal{F}[\xi]$ is numerically integrable.*

Proof. Consider the first layer. We can write the sigmoidal output of the $(l+1)^{\text{th}}$ layer as a function of the previous layer; that is,

$$\sigma^{(l+1)} = g\left(\int_{R^{(l)}} w^{(l)}(j_l, j_{l+1})\sigma^{(l)}(j_l)\, dj_l\right). \tag{3.1.1}$$

Clearly this composition can be expanded using the polynomial definition of the weight surface. Hence

$$\begin{aligned}\sigma^{(l+1)} &= g\left(\int_{R^{(l)}} \sigma^{(l)}(j_l)\sum_{x_{2l+1}}^{Z_Y^{(l)}}\sum_{x_{2l}}^{Z_X^{(l)}} k_{x_{2l},x_{2l+1}} j_l^{x_{2l}} j_{l+1}^{x_{2l+1}}\, dj_l\right)\\ &= g\left(\sum_{x_{2l+1}}^{Z_Y^{(l)}} j^{x_{2l+1}}\sum_{x_{2l}}^{Z_X^{(l)}} k_{x_{2l},x_{2l+1}}\int_{R^{(l)}}\sigma^{(l)}(j_l) j_l^{x_{2l}}\, dj_l\right),\end{aligned} \tag{3.1.2}$$

and therefore $\sigma^{(l+1)}$ is numerically integrable. For the purpose of constructing an algorithm, let $I^{(l)}_{x_{2l}}$ be the evaluation of the integral in the above definition for any given x_{2l}

It is important to note that the previous prioof requires that $\sigma^{(l)}$ be Riemann integrable. Hence, with ξ satisfying those conditions it follows that every $\sigma^{(l)}$ is integrable inductively. That is, because $\sigma^{(0)}$ is integrable it follows that by the numerical integrability of all l, $\mathcal{F}[\xi] = \sigma^{(L)}$ is numerically integrable. This completes the proof. □

Using the logic of the previous proof, it follows that the development of some inductive algorithm is possible.

3.1.1 Feed Forward Algorithm

Fortunately in the proof it was shoiwn that by calculation of a constant, I, on each layer, the functional neural network becomes numerically integrable. The mechanism by which this can occur leads us to a simple algorithm for calculating $\mathcal{F}[\xi]$:

1. For each $l \in \{0, \dots, L-1\}$

   (a) For all $t \in Z^{(l)}_X$, calculate

   $$I^{(l)}_t = \int_{R^{(l)}} \sigma^{(l)}(j_l) j^t_l \, dj_l. \tag{3.1.3}$$

   (b) Calculate, for every $s \in Z^{(l)}_Y$,

   $$C^{(l)}_s = \sum_{x_{2l}}^{Z^{(l)}_X} k_{x_{2l},s} I^{(l)}_{x_{2l}} \tag{3.1.4}$$

   (c) Finally, using (3.1.3) and (3.1.4), cache

   $$\sigma^{(l+1)} = g\left(\sum_{x_{2l+1}}^{Z^{(l)}_Y} j^{x_{2l+1}} C^{(l)}_{x_{2l+1}} \right) \tag{3.1.5}$$

   for use in the next iteration of loop.

2. The last $\sigma^{(l)}$ calculated is the output of the functional neural network.

3.2 Fast Error Backpropagation

The ability to propagate through the functional neural network has been defined. Now to be able to employ the network, a method of training is derived. Each weight coefficient in each weight function between layers needs to be updated according to the error of the output function. The theoretical method has already been defined in the paper but it has a time complexity of $O(n^{2L})$ for each weight coefficient, and it does not satisfy the numerical integration condition in this section. Thus in the following section we make claims as to the integrability of functional neural networks and thereafter their computational complexity.

We first propose the following lemma as to aid in our derivation of a computationally suitable error backpropagation algorithm.

Lemma 3.2.1. *Given some layer, $l > 0$, in $\mathcal{F}$, functions of the form $\Psi^{(l)} = g'\left(\Sigma_l \sigma^{(l)}\right)$ are numerically integrable.*

Proof. If

$$\Psi^{(l)} = g'\left(\int_{R^{(l-1)}} \sigma^{(l-1)} w^{(l-1)}\ dj_{l-1}\right) \tag{3.2.1}$$

then

$$\Psi^{(l)} = g'\left(\sum_{b}^{Z_Y^{(l-1)}} j_l^b \sum_{a}^{Z_X^{(l-1)}} k_{a,b}^{(l-1)} \int_{R^{(l-1)}} \sigma^{(l-1)} j_{l-1}^a\ dj_{l-2}\right) \tag{3.2.2}$$

hence Ψ can be numerically integrated and thereby evaluated. □

The ability to simplify the derivative of the output of each layer greatly reduces the computational time of the error backpropagation. It becomes a function defined on the interval of integration of the next iterated integral.

Theorem 3.2.2. *The gradient, $\nabla E(\gamma, \delta)$, for the error function (2.3.2) on some $\mathcal{F}$ can be evaluated numerically.*

Proof. Recall that E over $\mathcal{F}$ is composed of $k_{x,y}^{(l)}$ for $x \in Z_X^{(l)}, y \in Z_Y^{(l)}$, and $0 \leq l \leq L$. If we show that $\frac{\partial E}{\partial k_{x,y}^{(l)}}$ can be numerically evaluated for arbitrary, l, x, y, then every component of ∇E is numerically evaluable and hence ∇E can be numerically evaluated. Given some arbitrary l in $\mathcal{F}$, let $n = L - l$. We will examine the particular partial derivative for the case that $n = 1$, and then for arbitrary n, induct over each iterated integral.

Consider the following expansion for $n = 1$,

$$\begin{aligned}
\frac{\partial E}{\partial k_{x,y}^{(L-n)}} &= \frac{\partial}{\partial k_{x,y}^{(L-1)}} \frac{1}{2} \int_{R^{(l)}} \left[\mathcal{F}(\gamma) - \delta\right]^2\ dj_L \\
&= \int_{R^{(l)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} \int_{R^{(l-1)}} j_{L-1}^x j_L^y \sigma^{(L-1)} dj_{L-1}\ dj_L \\
&= \int_{R^{(l)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} j_L^y \int_{R^{(l-1)}} j_{L-1}^x \sigma^{(L-1)} dj_{L-1}\ dj_L
\end{aligned} \tag{3.2.3}$$

Since the second intgral in (3.2.3) is exactly $I_x^{(L-1)}$ from (3.1.3), it follows that

$$\frac{\partial E}{\partial k_{x,y}^{(n)}} = I_x^{(L-1)} \int_{R^{(l)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} j_L^y\ dj_L \tag{3.2.4}$$

and clearly for the case of $n = 1$, the theorem holds.

Now we will show that this is all the case for larger n. It will beconme clear why we have chosen to include $n = 1$ in the proof upon expansion of the pratial derivative in these higher order cases.

Let us expand the gradient for $n \in \{2, \ldots, L\}$.

$$\begin{aligned}
\frac{\partial E}{\partial k_{x,y}^{L-n}} = \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^{(L)} \underbrace{\int_{R^{(L-1)}} w^{(L-1)} \Psi^{(L-1)} \int \cdots \int_{R^{(L-n+1)}} w^{(L-n+1)} \Psi^{(L-n+1)}}_{n-1 \text{ iterated integrals}} \\
\int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^a j_{L-n+1}^b\ dj_{L-n} \ldots dj_L
\end{aligned} \tag{3.2.5}$$

As aforementioned, proving the $n = 1$ case is required because for $n = 1$, (3.2.5) has a section of $n - 1 = 0$ iterated integrals which cannot be possible for the proceeding logic.

We now use the order invariance propery of iterated integrals (that is, $\int_A \int_B f(x, y)\, dxdy = \int_B \int_A f(x, y)\, dydx$) and reverse the order of integration of (3.2.5).

In order to reverse the order of integration we must ensure each iterated integral has an integrand which contains variables which are guarenteed integration over some region. To examine this, we propose the following recurrence relation for the gradient.

Let $\{B_s\}$ be defined along $L - n \leq s \leq L$, as follows

$$\begin{aligned} B_L &= \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta]\, \Psi^{(L)} B_{L-1}\, dj_L, \\ B_s &= \int_{R^{(l)}} \Psi^{(l)} \sum_a^{Z_X^{(l)}} \sum_b^{Z_Y^{(l)}} j_l^a j_{l+1}^b B_{l-1}\, dj_l, \\ B_{L-n} &= \int_{R^{(L-n)}} j_{L-n}^x j_{L-n+1}^y\, dj_{L-n} \end{aligned} \tag{3.2.6}$$

such that $\frac{\partial E}{\partial k_{x,y}^{(l)}} = B_L$. If we wish to reverse the order of integration, we must find a reccurrence relation on a sequence, $\{\mathfrak{B}_s\}$ such that $\frac{\partial E}{\partial k_{x,y}^{(L-n)}} = \mathfrak{B}_{L-n} = B_L$. Consider the gradual reversal of (3.2.5).

Clearly,

$$\begin{aligned} \frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^x \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^L \int_{R^{(L-1)}} w^{(L-1)} \Psi^{(L-1)} \\ \int \cdots \int_{R^{(L-n+1)}} j_{L-n+1}^y w^{(L-n+1)} \Psi^{(L-n+1)}\, dj_{L-n+1} \ldots dj_L dj_{L-n} \end{aligned} \tag{3.2.7}$$

is the first order reversal of (3.2.5). We now show the second order case with first weight function expanded.

$$\begin{aligned} \frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^x \int_{R^{(L-n+1)}} \sum_b^{Z_Y} \sum_a^{Z_X} k_{a,b} j_{L-n+1}^{a+y} \Psi^{(L-n+1)} \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^L \\ \int \cdots \int_{R^{(L-n+1)}} j_{L-n+2}^b w^{(L-n+2)} \Psi^{(L-n+2)}\, dj_{L-n+1} \ldots dj_L dj_{L-n}. \end{aligned} \tag{3.2.8}$$

Repeated iteration of the method seen in (3.2.7) and (3.2.8), where the inner most integral is moved to the outside of the $(L - s)^{\text{th}}$ iterated integral, with s is the iteration, yields the following full reversal of (3.2.5). For notational simplcity recall that $l = L - n$, then

$$\begin{aligned} \frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(l)}} \sigma^{(l)} j_l^x \int_{R^{(l+1)}} \sum_a^{Z_X^{(l+1)}} j_{l+1}^{a+y} \Psi^{(l+1)} \int_{R^{(l+2)}} \sum_b^{Z_Y^{(l+1)}} \sum_c^{Z_X^{(l+2)}} k_{a,b}^{(l+1)} j_{l+2}^{b+c} \Psi^{(l+2)} \\ \int_{R^{(l+3)}} \sum_d^{Z_Y^{(l+2)}} \sum_e^{Z_X^{(l+3)}} k_{c,d}^{(l+2)} j_{l+3}^{d+e} \Psi^{(l+3)} \int \cdots \int_{R^{(L)}} \sum_q^{Z_Y^{(L-1)}} k_{p,q} j_L^q\, [\mathcal{F}(\gamma) - \delta] \Psi^{(L)} \\ dj_L \ldots dj_{L-n}. \end{aligned} \tag{3.2.9}$$

Observing the reversal in (3.2.9), we yield the following reccurence relation for $\{\mathfrak{A}_s\}$. Bare in mind, $l = L - n$, x and y still correspond with $\frac{\partial E}{\partial k^{(l)}_{x,y}}$, and the following relation uses its definition on s for cases not otherwise defined.

$$\begin{aligned}
\mathfrak{A}_{L,t} &= \int_{R^{(L)}} \sum_{b}^{Z_Y^{(L-1)}} k_{t,b} j_L^b \left[\mathcal{F}(\gamma) - \delta\right] \, dj_L. \\
\mathfrak{A}_{s,t} &= \int_{R^{(s)}} \sum_{b}^{Z_Y^{(s-1)}} \sum_{a}^{Z_X^{(s)}} j_s^{a+b} \Psi^{(s)} \mathfrak{A}_{s+1,a} \, dj_s. \\
\mathfrak{A}_{l+1} &= \int_{R^{(l+1)}} \sum_{a}^{Z_X^{(l+1)}} j_{l+1}^{a+y} \Psi^{(l+1)} \mathfrak{A}_{l+2,a} \, dj_{l+1}. \\
\frac{\partial E}{\partial k^{(l)}_{x,y}} = \mathfrak{A}_l &= \int_{R^{(l)}} j_l^x \sigma^{(l)} \mathfrak{A}_{l+1} \, dj_l.
\end{aligned} \tag{3.2.10}$$

Note that $\mathfrak{A}_{L-n} = B_L$ by this logic.

With (3.2.10), we need only show that $\mathfrak{A}_{L-n}$ is integrable. Hence we induct on $L - n \leq s \leq L$ over $\{\mathfrak{A}_s\}$ under the proposition that $\mathfrak{A}_s$ is not only numerically integrable but also constant.

Consider the base case $s = L$. For every t, because every function in the integrand of $\mathfrak{A}_L$ in (3.2.10) is composed of j_L, functions of the form $\mathfrak{A}_L$ must be numerically integrable and clearly, $\mathfrak{A}_L \in \mathbb{R}$.

Now suppose that $\mathfrak{A}_{s+1,t}$ is numerically integrable and constant. Then, trivially, $\mathfrak{A}_{s,u}$ is also numerically integrable by the contents of the integrand in (3.2.10) and $\mathfrak{A}_{s,u} \in \mathbb{R}$. Hence, the proposition that $s + 1$ implies s holds for $l + 1 < s < L$.

Lastly we must show that both $\mathfrak{A}_{l+1}$ and $\mathfrak{A}_l$ are numerically integrable. By induction $\mathfrak{A}_{l+2}$ must be numerically integrable. Hence by the contents of its integrand $\mathfrak{A}_{l+1}$ must also be numerically integrable and real. As a result, $\mathfrak{A}_l = \frac{\partial E}{\partial k^{(l)}_{x,y}}$ is real and numerically integrable.

Since we have shown that $\frac{\partial E}{\partial k^{(l)}_{x,y}}$ is numerically integrable, ∇E must therefore be numerically integrable as aforementioned. This completes the proof. □

3.2.1 Fast Error Backpropagation Algorithm

The complexity of the network has been greatly reduced by showing that the error with respect to each weight coefficient can be numerically integrable. The total time complexity of the algorithm however can be reduced even more through the implementation of caching. Many times during the calculations in error backpropagation, the same values are reexamined a redundant number of times. Caching tries to reduce the amount of information needed to be processed. The implementable algorithm for functional error backpropagation occurs as follows.

1. Calculate and cache all the Ψ^n for $1 \leq n \leq L$
2. Calculate $\frac{\partial E}{\partial k^0_{x,y}}$. In order to do so, calculate and cache all of the λ for the layers starting at L and going to 1. Finally calculate $\frac{\partial E}{\partial k^0_{x,y}} = \int_{R_0} \sigma^0 j_0^x \lambda^0_{x_1} d_{j_0}$.

3. Calculate $\frac{\partial E}{\partial k^1_{x,y}}$. The λ can be reused from $\lambda^2_{x_5}...\lambda^{L-1}_{x_{2L-1}}$. Calculate and cache $\int_{R^2}\sum_{x_5}^{Z^2} j_2^{x_5}\Psi^2 j_2^{x_4} d_{j_2}\lambda^2_{x_5} = \lambda^1_{x_3}$. Finally you can calculate $\frac{\partial E}{\partial k^1_{x,y}} = \int_{R_1}\sigma^1 j_1^x \lambda^1_{x_3} d_{j_1}$.

4. Continue calculating $\frac{\partial E}{\partial k_{x,y}}$ for the rest of the layers similar to step 3. For each successive, all the λ can be reused except for the initial two. However, because each λ determines the previous λ, only the first λ needs to be reused. For layer l, use $\lambda^{2l-2}_{x_{2l-3}}$.

5. Calculate $\frac{\partial E}{\partial k^{L-1}_{x,y}}$ as normal. However, this layer does not use any of the cached λ and is directly calculated.

3.3 Computational Complexity

As aforementioned, one would imagine that the iterated recursion relation defining functional neural networks would require immense computational complexity. This follows namely from the theory of elementary integration; that is, for any multi layer functional neural network, elementary integration is not possible unless the sigmoidal activation is a linear rectifier. Hence numerical integration on the last layer would require evaluation of the previous layers multiple times, which in turn would require the numerical evaluation of each anterior layer. This is a process which clearly would require a computational complexity proportional to the number of layers. This time requirement manifests itself in both the feed forward and back propagation actions for all $\mathcal{F}$. To prevent this difficulty we will examine how the formulation of the Fast Feed-forward and Backpropagation algorithms affect time complexity.

3.3.1 Forward Propagation

Consider the feed-forward action. Normally the evaluation of $\mathcal{F}[\xi]$ at any point j_L requires that each layer on the anterior be evaluated. To see how computationally taxing this is, recall the complexity of the second order quadrature algorithms. Simpson's quadrature requires $O(pe(f))$ where $e(f)$ is the cost of evaluating any given function f at a point j along the interval P which is divided into p equal partitions. Now consider that evaluating each layer of the neural network also requires the evaluation of the weight polynomial which takes $O(Z_X^{(l)} Z_Y^{(l)})$. For the sake of simplicity we subdivide our integration interval into as many parts as there are polynomial coefficients in one direction. Hence, the evaluation of one layer is $O(Z_x^3 e(\sigma_{l-1}))$. Clearly this process will compound exponentially leading us to the final complexity for network evaluation; that is,

$$O(\mathcal{F}) = O\left(\prod_{l=0}^{L} Z_l^3\right). \tag{3.3.1}$$

However with the addition of the fast feed-forward algorithm, the evaluation of each σ is cached, and hence evaluation becomes nominally, $O(X_l^2)$. Hence the evaluation of the functional neural networks on the last layer requires caching which yields

$$O(\mathcal{F}_1) = O\left(\sum_{l}^{L} Z_l^3 Z_{l-1}^2\right). \tag{3.3.2}$$

where $\mathcal{F}_1$ is the first evaluation, after which computational complexity simply becomes

$$O(\mathcal{F}) = O\left(Z_L^2\right). \tag{3.3.3}$$

3.3.2 Backpropagation

The case of fast error back propagation is slightly more difficult as the complexity of one iteration hinges on the number of layers for some $\mathcal{F}$. For all intensive purposes, the calculation of weight change on any one iteration requires the calculation of every Ψ^l, a process taking $O(l)$. After the calculation of each Ψ an evaluation of the gradient for every coefficient on the first layer must occur and hence every λ must be calculated. Calculation of λ_L requires each preceding λ, and hence the process takes

$$O\left(Z_X^{(0)} Z_Y^{(0)} \sum_{l}^{L} Z_l^3 Z_{l-1}^2\right). \tag{3.3.4}$$

Finally, because the calculation of each subsequent gradient requires one less λ and each λ_t can be evaluated with $O(1)$ access complexity, the above complexity dominates in the calculation. Therefore, the use of fast error back-propagation is polynomial whereas direct computation is exponential.

4 Conclusion

When the research question was posed, the goal was to develop a mathematical construct that is continuously homologous to standard and discrete artificial neural networks. This along with inspiration from common economic techniques like interpolation led us to consider a neural network structure containing infinite input, hidden, and output neurons. This new construction is named the functional neural network and has the same properties as discrete neural networks.

The proposition of functional neural networks lead to new insights into the black box model of traditional neural networks. First, functional networks are a logical generalization of the discrete neural network and therefore all theorems shown for traditional neural networks apply to piecewise functional neural networks. Furthermore the creation of homologous theorems for universal approximation provided a way to find a relationship between the weights of traditional neural networks. This suggests that the discrete weights of a normal artificial neural network can be transformed into continuous surfaces which approximate kernels satisfying the training dataset. Functional neural networks are also able to approximate bounded linear operators (the general form of functions), homologous to how ANNs apprxoimate functions. Finally a continuous version of the error backpropagation algorithm was developed, providing information into how the discrete error backprop algorithm operates: the chain rule just becomes convolution integration across partially derived anterior layers.

These algorithms, while theoretically feasible, were originally too computationally complex to have a practical application. The algorithms were reapproached mathematically to make feed forward and error backpropagation numerically integrable. This property is shown through the expansion of the weight polynomial and the interchanging of iterated integrals. From this, a computational implementation of the new generalized construct, FNNs, is created. Furthermore, the process of caching important values in weight coefficient calculation greatly reduced the original factorial complexity such that the algorithms could run in reasonable computational time.

Thus, the functional neural network not only provides novel mathematical insights behind the traditional neural network algorithm, it also becomes a feasible new machine learning method. This new field has many potential directions especially with continued techniques from mathematical analysis.

4.1 Future Work

The computational algorithm was applied to analyze the Laplace transform. However, there is still a range of applications to be explored. The original intent of this paper was to explore continuous data that exists in the real world. Sound waves and sight analysis by ears and eyes occurs on a continuous level yet most of the data analysis in these fields has discretized the data for numerical evaluation. The insertion of continuous sinusoidal definition of these datasets into a FNN would be a possible field of exploration.

On the theoretical side, it must be proven, or disproven, that functional neural networks are or are not analytically integrable in closed form. If they are integrable then the aforementioned computational implementation will be very simple and possibly faster than discrete neural networks. In the case that they are not, functional neural networks may remain simply a theoretical construct for understanding discrete neural networks. Evidence has been shown that it is integrable using a linear sigmoid activation functions. However,

for nonlinear sigmoid activation functions, there is a strong indication that there is no closed form interval solution.