Generalized Artifical Neural Networks

Approximated by Weierstrass Polynomials

Guss, William
wguss@berkeley.edu

April 5, 2016

Abstract

NOTE: This is an old abstract for only the first part of the paper (FNNs).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

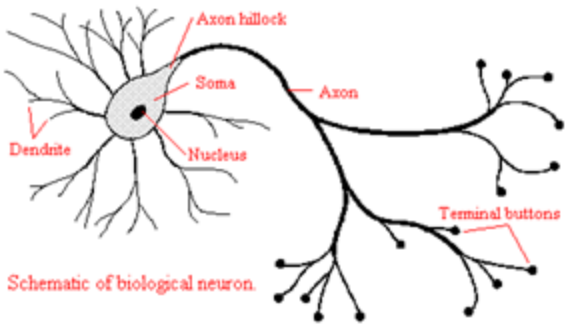

Figure 1: A biological neuron neuralimage.

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*

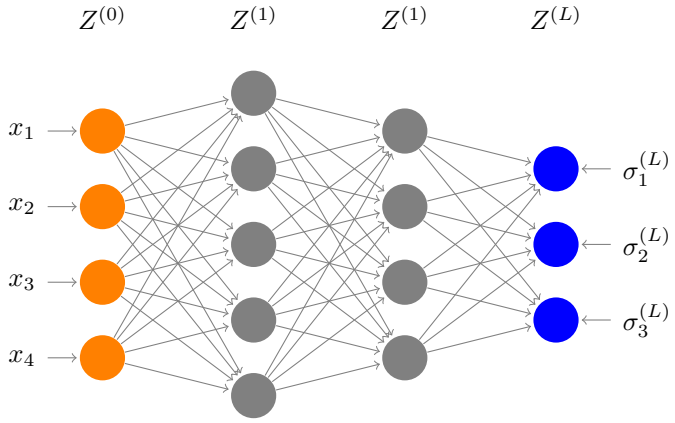

Figure 2: An example of a feed-forward ANN, $\mathcal{N}$ with four layers.

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 layersvisualcortex.

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 $\mathcal{N}$.

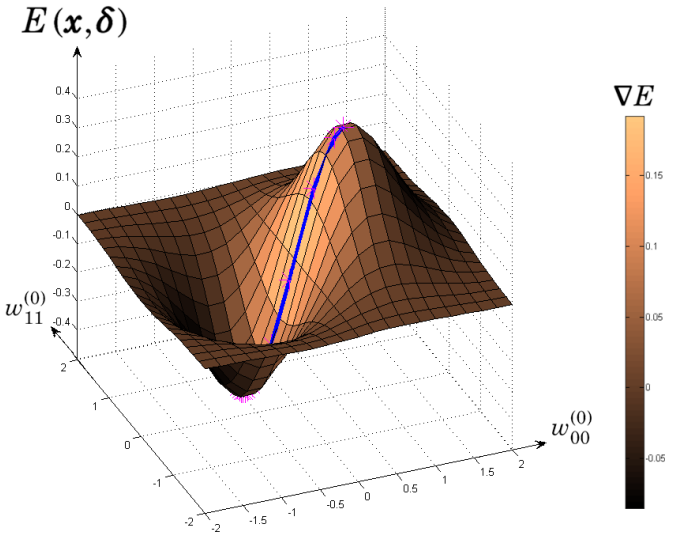

Figure 3: Gradient descent on $E(\boldsymbol{x},\boldsymbol{\delta})$ with descent path shown in blue.

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

2.1 Derivation

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 while caching previous computations along the way. Likewise, for the error back-propagation method, we will cache and eliminate variables until we are able to define an algorithm that is suitably computable.

In order to begin exploration of such an implementation, we need to both more rigorously define our notion of the weight surface and what is itself computationally evaluable.

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

$$w^{(l)}(i,j) = \sum_{x_{2l+1}}^{Z_Y^{(l)}} \sum_{x_{2l}}^{Z_X^{(l)}} k^{(l)}_{x_{2l},x_{2l+1}} j_l^{x_{2l}} j_{l+1}^{x_{2l+1}} \tag{2.1.1}$$

where $k_{a,b}$ is some real valued coefficient and the order of the polynomial is arbitrary.

With that in mind, let us define the notion of numerical integrability (and thereby evaluability).

Definition 2.1.2. *Let $f : \mathbb{R}^{2n} \to \mathbb{R}$, We say that some function ϕ of the form*

$$\phi(x) = \int_{E \subset \mathbb{R}^n} f(x,y)dy \tag{2.1.2}$$

can be numerical integrated if and only if

$$\phi(x) = h(x) \int_{E \subset \mathbb{R}^n} g(y)dy \tag{2.1.3}$$

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

Using the above definition we will explore the feed-forward and back-propagation actions of any given $\mathcal{F}$.

2.1.1 Feed-Forward Propagation

Consider the notion of numerical integrability defined in definition **??** applied to the calculation of output for some functional neural network.

Theorem 2.1.3. *If $\mathcal{F}$ is a functional neural network with L consecutive layers, then given any l such that $0 \leq l < L$, $\sigma^{(l+1)}$ 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,

$$\sigma^{(l+1)} = g\left(\int_{R^{(l)}} w^{(l)}(j_l, j_{l+1})\sigma^{(l)}(j_l)\, dj_l\right). \tag{2.1.4}$$

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

$$\begin{aligned}\sigma^{(l+1)} &= g\left(\int_{R^{(l)}} \sigma^{(l)}(j_l) \sum_{x_{2l+1}}^{Z_Y^{(l)}} \sum_{x_{2l}}^{Z_X^{(l)}} k_{x_{2l},x_{2l+1}} j_l^{x_{2l}} j_{l+1}^{x_{2l+1}}\, dj_l\right) \\ &= g\left(\sum_{x_{2l+1}}^{Z_Y^{(l)}} j^{x_{2l+1}} \sum_{x_{2l}}^{Z_X^{(l)}} k_{x_{2l},x_{2l+1}} \int_{R^{(l)}} \sigma^{(l)}(j_l) j_l^{x_{2l}}\, dj_l\right),\end{aligned} \tag{2.1.5}$$

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

It is important to note that the previous proof 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.

2.1.2 Feed-Forward Algorithm

Fortunately in the proof it was shown 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 $l \in \{0, \ldots, L-1\}$
   (a) For all $t \in Z_X^{(l)}$, calculate
   $$I_t^{(l)} = \int_{R^{(l)}} \sigma^{(l)}(j_l) j_l^t\, dj_l. \tag{2.1.6}$$
   (b) Calculate, for every $s \in Z_Y^{(l)}$,
   $$C_s^{(l)} = \sum_{x_{2l}}^{Z_X^{(l)}} k_{x_{2l},s} I_{x_{2l}}^{(l)} \tag{2.1.7}$$
   (c) Finally, using (**??**) and (**??**), cache
   $$\sigma^{(l+1)} = g\left(\sum_{x_{2l+1}}^{Z_Y^{(l)}} j^{x_{2l+1}} C_{x_{2l+1}}^{(l)}\right) \tag{2.1.8}$$
   for use in the next iteration of loop.
2. The last $\sigma^{(l)}$ calculated is the output of the functional neural network.

2.1.3 Continuous Error Backpropagation

Just as important as the feed-forward of neural network algorithms is the notion of training. As is common with many non-convex problems with discretized neural networks, a stochastic gradient descent method will be developed using a continuous analogue to error backpropagation.

As is typical in optimization, a loss function is defined as follows.

Definition 2.1.4. *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*

$$E = \frac{1}{2}\int_{R^{(L)}} (\mathcal{F}(\gamma_n) - \delta_n)^2 \; dj_L \tag{2.1.9}$$

This error definition follows from $\mathcal{N}$ as the typical error function for $\mathcal{N}$ is just the square norm of the difference of the desired and predicted output vectors. In this case we use the L^2 norm on $C(R^{(L)})$ in the same fashion.

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

Lemma 2.1.5. *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

$$\Psi^{(l)} = g'\left(\int_{R^{(l-1)}} \sigma^{(l-1)} w^{(l-1)} \; dj_{l-1}\right) \tag{2.1.10}$$

then

$$\Psi^{(l)} = g'\left(\sum_{b}^{Z_Y^{(l-1)}} j_l^b \sum_{a}^{Z_X^{(l-1)}} k_{a,b}^{(l-1)} \int_{R^{(l-1)}} \sigma^{(l-1)} j_{l-1}^a \; dj_{l-2}\right) \tag{2.1.11}$$

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 2.1.6. *The gradient, $\nabla E(\gamma, \delta)$, for the error function (**??**) 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 $n = 1$,

$$\begin{aligned}
\frac{\partial E}{\partial k_{x,y}^{(L-n)}} &= \frac{\partial}{\partial k_{x,y}^{(L-1)}} \frac{1}{2} \int_{R^{(l)}} [\mathcal{F}(\gamma) - \delta]^2 \; dj_L \\
&= \int_{R^{(l)}} [\mathcal{F}(\gamma) - \delta] \, \Psi^{(L)} \int_{R^{(l-1)}} j_{L-1}^x j_L^y \sigma^{(L-1)} dj_{L-1} \; dj_L \\
&= \int_{R^{(l)}} [\mathcal{F}(\gamma) - \delta] \, \Psi^{(L)} j_L^y \int_{R^{(l-1)}} j_{L-1}^x \sigma^{(L-1)} dj_{L-1} \; dj_L
\end{aligned} \tag{2.1.12}$$

Since the second integral in (**??**) is exactly $I_x^{(L-1)}$ from (**??**), it follows that

$$\frac{\partial E}{\partial k_{x,y}^{(n)}} = I_x^{(L-1)} \int_{R^{(l)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} j_L^y \, dj_L \tag{2.1.13}$$

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 become 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 $n \in \{2, \dots, L\}$.

$$\frac{\partial E}{\partial k_{x,y}^{L-n}} = \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^{(L)} \underbrace{\int_{R^{(L-1)}} w^{(L-1)} \Psi^{(L-1)} \int \cdots \int_{R^{(L-n+1)}} w^{(L-n+1)} \Psi^{(L-n+1)}}_{n-1 \text{ iterated integrals}}$$
$$\int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^a j_{L-n+1}^b \, dj_{L-n} \dots dj_L \tag{2.1.14}$$

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

We now use the order invariance properly 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 (**??**).

In order to reverse the order of integration we must ensure each iterated integral has an integrand which contains variables which are guaranteed 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

$$\begin{aligned} B_L &= \int_{R^{(L)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} B_{L-1} \, dj_L, \\ B_s &= \int_{R^{(l)}} \Psi^{(l)} \sum_a^{Z_X^{(l)}} \sum_b^{Z_Y^{(l)}} j_l^a j_{l+1}^b B_{l-1} \, dj_l, \\ B_{L-n} &= \int_{R^{(L-n)}} j_{L-n}^x j_{L-n+1}^y \, dj_{L-n} \end{aligned} \tag{2.1.15}$$

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 reoccurrence 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 (**??**).

Clearly,

$$\begin{aligned} \frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^x \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^L \int_{R^{(L-1)}} w^{(L-1)} \Psi^{(L-1)} \\ \int \cdots \int_{R^{(L-n+1)}} j_{L-n+1}^y w^{(L-n+1)} \Psi^{(L-n+1)} \, dj_{L-n+1} \dots dj_L dj_{L-n} \end{aligned} \tag{2.1.16}$$

is the first order reversal of (**??**). We now show the second order case with first weight

function expanded.

$$\frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(L-n)}} \sigma^{(L-n)} j_{L-n}^{x} \int_{R^{(L-n+1)}} \sum_{b}^{Z_Y} \sum_{a}^{Z_X} k_{a,b} j_{L-n+1}^{a+y} \Psi^{(L-n+1)} \int_{R^{(L)}} [\mathcal{F}(\gamma) - \delta] \Psi^{L} \int \cdots \int_{R^{(L-n+1)}} j_{L-n+2}^{b} w^{(L-n+2)} \Psi^{(L-n+2)} \, dj_{L-n+1} \dots dj_L dj_{L-n}. \tag{2.1.17}$$

Repeated iteration of the method seen in (**??**) and (**??**), 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 (**??**). For notational simplicity recall that $l = L - n$, then

$$\frac{\partial E}{\partial k_{x,y}^{(l)}} = \int_{R^{(l)}} \sigma^{(l)} j_{l}^{x} \int_{R^{(l+1)}} \sum_{a}^{Z_X^{(l+1)}} j_{l+1}^{a+y} \Psi^{(l+1)} \int_{R^{(l+2)}} \sum_{b}^{Z_Y^{(l+1)}} \sum_{c}^{Z_X^{(l+2)}} k_{a,b}^{(l+1)} j_{l+2}^{b+c} \Psi^{(l+2)} \int_{R^{(l+3)}} \sum_{d}^{Z_Y^{(l+2)}} \sum_{e}^{Z_X^{(l+3)}} k_{c,d}^{(l+2)} j_{l+3}^{d+e} \Psi^{(l+3)} \int \cdots \int_{R^{(L)}} \sum_{q}^{Z_Y^{(L-1)}} k_{p,q}^{(L-1)} j_{L}^{q} \, [\mathcal{F}(\gamma) - \delta] \Psi^{(L)} \, dj_L \dots dj_{L-n}. \tag{2.1.18}$$

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

$$\begin{aligned} \mathfrak{A}_{L,t} &= \int_{R^{(L)}} \sum_{b}^{Z_Y^{(L-1)}} k_{t,b}^{(L-1)} j_{L}^{b} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} \, dj_L. \\ \mathfrak{A}_{s,t} &= \int_{R^{(s)}} \sum_{b}^{Z_Y^{(s-1)}} \sum_{a}^{Z_X^{(s)}} k_{t,b}^{(s-1)} j_{s}^{a+b} \Psi^{(s)} \mathfrak{A}_{s+1,a} \, dj_s. \\ \mathfrak{A}_{l+1} &= \int_{R^{(l+1)}} \sum_{a}^{Z_X^{(l+1)}} j_{l+1}^{a+y} \Psi^{(l+1)} \mathfrak{A}_{l+2,a} \, dj_{l+1}. \\ \frac{\partial E}{\partial k_{x,y}^{(l)}} = \mathfrak{A}_l &= \int_{R^{(l)}} j_{l}^{x} \sigma^{(l)} \mathfrak{A}_{l+1} \, dj_l. \end{aligned} \tag{2.1.19}$$

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

With (**??**), 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 (**??**) 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 (**??**) 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, $\mathfrak{A}_l = \frac{\partial E}{\partial k_{x,y}^{(l)}}$ is real and numerically integrable.

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

2.1.4 Continuous Error Backpropagation Algorithm

The logic of the proceeding proof required the establishment of $\{\mathfrak{A}_s\}$ as a sequence with a recurrence relation defining each component of the gradient on some coefficient in the network. Interestingly enough for L large enough certain elements of $\{\mathfrak{A}_s\}$ can be reused for different n. In order to more accurately describe this process, we propose the following algorithm for calculating gradients in stochastic gradient descent.

1. For all $\lambda \in \{1, \ldots, L\}$, calculate $\Psi^{(\lambda)}$.
2. Calculate $K - \nabla E \to K$ where K is the vector of all coefficient matrices by calculating each partial matrix over $l \in \{0, \ldots, L-1\}$
   (a) If $l = L - 1 \Leftrightarrow (n = 1)$, then store into the coefficient matrix
   $$k_{x,y}^{(L-1)} - I_x^{(L-1)} \int_{R^{(L)}} \left[\mathcal{F}(\gamma) - \delta\right] \Psi^{(L)} j_L^y \, dj_L \to k_{x,y}^{(L-1)} \tag{2.1.20}$$
   for every x, y.
   (b) If $l < L - 1 \Leftrightarrow n > 1$, then
      i. Ensure that $\mathfrak{A}_{l+2,t}$ from (**??**) is computed for all $t \in Z_X^{(l+1)}$
      ii. Then for all $x \in Z_X^{(l)}$ and $y \in Z_Y^{(l)}$,
         A. Compute
         $$\mathfrak{A}_{l+1} = \int_{R^{(l+1)}} \sum_a^{Z_X^{(l+1)}} j_{l+1}^{a+y} \Psi^{(l+1)} \mathfrak{A}_{l+2,a} \, dj_{l+1}. \tag{2.1.21}$$
         B. Compute
         $$\frac{\partial E}{\partial k_{x,y}^{(l)}} = \mathfrak{A}_l = \int_{R^{(l)}} j_l^x \sigma^{(l)} \mathfrak{A}_{l+1} \, dj_l. \tag{2.1.22}$$
         C. Update the weights such that
         $$k_{x,y}^{(l)} - \mathfrak{A}_l \to k_{x,y}^{(l)} \tag{2.1.23}$$

With the completion of the implementation, the theoretical definition of functional neural networks is complete.

3 Generalized Artificial Neural Networks

We have clearly demonstrated the theoretical power of FNNs, but does there exist some more general structure which includes FNNs and ANNs without piecewise approximation? In other words, can we determine a form which encompasses the discrete classification of continuous datasets or visa versa all the while maintains either the power of discrete or functional artificial neural networks? The solution to such questions must furthermore maintain universal approximation and computational evaluability so as to allow the implementation of a real-world algorithm.

In this section of the paper we will propose the Generalized Artificial Neural Network through the examination of specific operations on different layers.

3.1 The Generalization of ANNs

In order to generalize ANNs in a way that follows logically, we take the initial definition of functional neural networks and consider the notion of a layer.

Definition 3.1.1. *If A, B are (possibly distinct) Hilbert spaces over $\mathbb{R}$, we say $\mathcal{G} : A \to B$ is a generalized neural network if and only if*

$$\begin{aligned} \mathcal{G} : \sigma^{(l+1)} &= g\left(T_l\left[\sigma^{(l)}\right] + \beta^{(l)}\right) \\ \sigma^{(0)} &= \xi \end{aligned} \tag{3.1.1}$$

for some input $\xi \in A$.

In (**??**), it is unclear how the form of T_l is restricted. It might be recalled that T_l takes a form similar to the operation of a layer $l+1$ on l in the proof of universal approximation for $\{\mathcal{F}\}$. Let us consider a few other such forms of T_l by first stating a formal definition.

Definition 3.1.2. *We say that T_l is the operation of a layer $l+1$ on l in some $\mathcal{G}$ if and only if for $l = L-1$, $T : C \to B$ with C the codomain of $\sigma^{(0)}$ and for $l = 0$, $T_l : A \to D$ where D is the domain of $\sigma^{(1)}$.*

Using the above definition it is now possible to construct different classes of T_l using ideas directly from the constructions of $\mathcal{N}$ and $\mathcal{F}$.

Definition 3.1.3. *We suggest several classes of T_l as follows*

- *T_l is said to be $\mathfrak{f}$ functional if and only if =*

$$\begin{aligned} T_l = \mathfrak{f} : C(R^{(l)}) &\to C(R^{(l+1)}) \\ \sigma &\mapsto \int_{R^{(l)}} \sigma(i) w^{(l)}(i,j)\ di. \end{aligned} \tag{3.1.2}$$

- *T_l is said to be $\mathfrak{n}$ discrete if and only if*

$$\begin{aligned} T_l = \mathfrak{n} : \mathbb{R}^n &\to \mathbb{R}^m \\ \vec{\sigma} &\mapsto \sum_j^m \vec{e}_j \sum_i^n \sigma_i w_{ij}^{(l)} \end{aligned} \tag{3.1.3}$$

  where $\vec{e}_j$ denotes the j^{th} basis vector in $\mathbb{R}^m$.

- *T_l is said to be $\mathfrak{n}_1$ transitional if and only if*

$$\begin{aligned} T_l = \mathfrak{n}_1 : \mathbb{R}^n &\to C(R^{(l+1)}) \\ \vec{\sigma} &\mapsto \sum_i^n \sigma_i w_i^{(l)}(j). \end{aligned} \tag{3.1.4}$$

- *T_l is said to be $\mathfrak{n}_2$ transitional if and only if*

$$\begin{aligned} T_l = \mathfrak{n}_2 : C(R^{(l)}) &\to \mathbb{R}^m \\ \sigma(i) &\mapsto \sum_j^m \vec{e}_j \int_{R^{(l)}} \sigma(i) w_j^{(l)}(i)\ di \end{aligned} \tag{3.1.5}$$

With the characterization of these layer classes complete, foundational theorems about this new generalization must be proposed. First we will show inclusion of other neural algorithms followed by universal approximation theorems (as by necessity).

Theorem 3.1.4. *If $\{\mathcal{G}\}$ is the set of all generalized artificial neural networks then $\{\mathcal{F}\}\cup\{\mathcal{N}\}$ is a subset of $\{G\}$.*

Proof. Since $\{\mathcal{N}\} \subset \{\mathcal{F}\}$, we need only show that $\{\mathcal{F}\} \subset \{\mathcal{G}\}$, but for the sake of the reader, consider that one might take any $\mathcal{N}$ and show equivalency with some $\mathcal{G}$ strictly composed of T_l which are discrete.

For the case of functional inclusion, consider any functional neural network $\mathcal{F}$ with L layers. Then let us construct $\mathcal{G} : C\left(R^{(0)}\right) \to C\left(R^{(L)}\right)$. First we endow $\mathcal{G}$ with T_l which are strictly functional. Let each T_l have an equivalent weight function $w^{(l)}$ to that of $\mathcal{F}$ on layer l. This construction yields the following recurrence relation:

$$\begin{aligned} \mathcal{G} : \sigma^{(l+1)}(j) &= g\left(\int_{R^{(l)}} \sigma^{(l)}(i) w^{(l)}(i,j)\ di + \beta^{(l)}\right) \\ \sigma^{(0)}(j) \quad &= \xi(j), \end{aligned} \tag{3.1.6}$$

which is exactly equivalent to that of $\mathcal{F}$. Hence for any $\mathcal{F}$ there exists a corresponding $\mathcal{G}$ with equivalency. Thus $\{\mathcal{F}\} \subset \{\mathcal{G}\}$. □

3.2 Input Quality as a Kernal Predicate for Neural Networks

In this section we will explore various anylitically integrable input functions on certain $\mathcal{G}$. In particular, we wish to explore the case of $\mathfrak{n}_2$ as it seems this construction may embody the most practical application of this new generalization: the classification of continuous signals. We wish to prove mathemnatically and demonstrate through application that such a method is better than current classifciation methods for continuous signals, or at the least, the two methods are isomorphic.

It is well established that all processes are sampled discretely, so in the least, some sort of approximate method must be used to create suitable inputs ξ for $\mathfrak{n}_2$ transitional layers. The following exploration is interesting.

Theorem 3.2.1. *Let $\mathcal{G}$ be a GANN with only one $\mathfrak{n}_2$ transitional layer. If a continuous function, say $f(t)$ is sampled uniformly from $t = 0$, to $t = N$, such that $x_n = f(n)$, and if $\mathcal{G}$ has an input function which is piecewise linear with*

$$\xi = (x_{n+1} - x_n)(z - n) + x_n \tag{3.2.1}$$

for $n \leq z < n+1$, then there exist some discrete neural network $\mathcal{N}$ such that $\mathcal{G}(\xi) = \mathcal{N}(\boldsymbol{x})$.

Proof. Recall that for the jth output neuron of a single layer discretized neural network,

$$\mathcal{N}_j(\boldsymbol{x}) = g\left(\sum_{i=1}^{N} w_{i,j}x_i + \beta\right). \tag{3.2.2}$$

Let this $\mathcal{N}$ compose the same sigmoid as the aforementioned $\mathfrak{n}_2$ transitional layer. We need only show that quivalence holds on the inside.

Because

$$\mathcal{G}_j(\xi) = g\left(\mathfrak{n}_2[\xi] + \beta\right), \tag{3.2.3}$$

it follows that,

$$\begin{aligned}
\mathfrak{n}_2(\xi) &= \int_R \xi(t)\, u_j(t)\, dt \\
&= \sum_{n}^{N-1} \int_n^{n+1} \left((x_{n+1} - x_n)(z-n) + x_n\right) u_j(z)\, dz \\
&= \sum_{n}^{N-1} (x_{n+1} - x_n) \int_n^{n+1} (z-n) \sum_{m}^{M} k_{m,j} z^m\, dz + x_n \int_n^{n+1} u_j(z)\, dz \\
&= \sum_{n}^{N-1} (x_{n+1} - x_n) \sum_{m}^{M} k_{m,j} \left[\frac{1}{m+2} z^{m+2} - \frac{n z^{m+1}}{m+1}\right]_n^{n+1} + \sum_{m}^{M} k_{m,j} x_n \frac{1}{m+1} z^{m+1}\Big|_n^{n+1}
\end{aligned} \tag{3.2.4}$$

Now, let $V_{n,j} = \sum_m k_{m,j} \left[\frac{1}{m+2} z^{m+2} - \frac{n z^{m+1}}{m+1}\right]_n^{n+1}$ and $Q_{n,j} = \sum_m k_{m,j} \frac{1}{m+1} z^{m+1}\big|_n^{n+1}$. We can now easily simplify (**??**) using the telescoping trick of summation. It follows that,

$$\mathfrak{n}_2(\xi) = x_N V_{N-1,j} + \sum_{n=2}^{N-1} x_n \left(Q_{n,j} - V_{n,j} + V_{n-1,j}\right) + x_1 \left(Q_{1,j} - V1, j\right). \tag{3.2.5}$$

By simply letting $w_{1,j} = \left(Q_{1,j} - V_{1,j}\right), w_{n,j} = \left(Q_{n,j} - V_{n,j} + V_{n-1,j}\right)$, and $w_{N,j} = V_{N-1,j}$ the following relation is satisfied, $\mathfrak{n}_2(\xi) = \mathfrak{n}(\boldsymbol{x})$. Hence, $\mathcal{G}(\xi) = \mathcal{N}(\boldsymbol{x})$ and the proof is complete. □

The previous theorem shows that at the least there is an isomorphism from $\{\mathcal{G}\}$ of that kind to the discretized neural network for piecewise linear interpolations. What can be said about higher order interpolations? The following theorem provides the same isomorphism.

Theorem 3.2.2. *Let $\phi : \mathbb{R}^N \to C(\mathbb{R})$ be the mapping for the polynomial interpolation of a uniformly sampled function $f(t)$ described by some vector of points. Then if $\mathcal{G}$ is a GANN with only one $\mathfrak{n}_2$ transitional layer, then there exists a discretized neural network $\mathcal{N}$ such that for all $\boldsymbol{x} \in \mathbb{R}^N$,*

$$\mathcal{G}\left[\phi(\boldsymbol{x})\right] = \mathcal{N}(\boldsymbol{x}). \tag{3.2.6}$$

Proof. It has been well established that a closed form definition for polynomial interpolation is as follows. If N points are sampled from a function $f(t)$ along intervals $[p_i, p_{i+1}]$,

$$\phi(\boldsymbol{x}) = \sum_{n=0}^{N} \left(\prod_{\substack{0 \le m \le n \\ m \neq n}} \frac{z - p_m}{p_n - p_m} \right) x_n. \tag{3.2.7}$$

For simplicity, let $h_n(z) = \prod_m \frac{z-p_m}{p_n-p_m} = \sum_s H_{n,s} z^s$ for some $H_{n,s}$ which satisfy the relationship. Because (**??**) implies that only equivalence of $\mathfrak{n}_2$ and $\mathfrak{n}$ must be shown,

$$\begin{aligned} \mathfrak{n}_2[\phi(\boldsymbol{x})] &= \int_R \phi(\boldsymbol{x})\ u_j\ dz \\ &= \sum_n^N x_n \int_R h_n(z) u_j(z)\ dz \\ &= \sum_n^N x_n \sum_s^N H_{n,s} \sum_t^T k_{t,j} \frac{1}{s+t+1} z^{s+t+1} \Big|_R \end{aligned} \tag{3.2.8}$$

Now defining $w_{n,j} = \sum_s^N H_{n,s} \sum_t^T k_{t,j} \frac{1}{s+t+1} z^{s+t+1}\big|_R$, it holds that $\mathfrak{n}_2[\phi(\boldsymbol{x})] = \mathfrak{n}[\boldsymbol{x}]$. Therefore, $\mathcal{G}[\phi(\boldsymbol{x})] = \mathcal{N}(\boldsymbol{x})$. This completes the proof.

□

Considering the logic of the previous proof in reverse, it is clear the some neural networks might infact be approximating these $\mathfrak{n}_2$ transitional layers. The following is a theorem that suggests some discrete neural networks approximate kernels on infinite dimensional space.

Theorem 3.2.3. *Discrete nueral networks with one hidden layer can approximate a kernel $K(x, w)$ which induces the inner product on the infinite dimensional L_2 space of continuous functions.*

Proof. Consider the kernel, $K(x, w) = \langle \phi(x), \phi(w) \rangle_{L_2}$ where $\phi : \mathbb{R}^N \to L_2(\mathbb{R})$ as in (**??**). □

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.