Mathematical Principles for Deconstructing Deep Learning: Theory and Application to Electromagnetic Signals

by

Evan Witz

A Dissertation

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Mathematical Sciences

August 4, 2022

APPROVED:

Professor Randy C. Paffenroth Worcester Polytechnic Institute Advisor	Professor Jian Zou Worcester Polytechnic Institute Committee Member
Professor Oren Mangoubi Worcester Polytechnic Institute Committee Member	Patrick Bidigare Synoptic Engineering External Committee Member
Professor Donald R. Brown Worcester Polytechnic Institute Committee Member	

Abstract

In the field of deep learning, there can often exist a divide between the theoretical and that of the applied. In this work, we attempt to bridge that gap by developing a theoretical framework for studying neural networks of a certain architecture, which can then be applied in the construction of real networks, which we apply to data coming from electromagnetic signal processing. These networks combine the best performance properties of more standard architectures with a beautifully rich algebraic structure. The theoretical portion of this work begins by studying the properties of networks with polynomial activation. We prove here a theorem demonstrating an algebraic decomposition of what we call Kronecker networks, as well as a concrete path toward a Universal Approximation Theorem for polynomial networks. In the applied portion, our focus is on the study of range localization of over-water electromagnetic signals. We attempt to use a received signal and deep learning to determine the distance from which the signal was sent. We explore the various atmospheric phenomena that can influence the signal and give a comparison to a statistical model, the maximum likelihood estimator. Finally, we bridge the two portions of this work by demonstrating that a network architecture inspired by our polynomial activation can achieve similar performance to a more standard architecture while enjoying a rich algebraic structure.

Acknowledgements

First and foremost, I'd like to express my endless gratitude towards my advisor, Randy Paffenroth of WPI. I came to Randy in late 2019 in the darkest part of my PhD journey – lost, advisorless, and nearly out of the program entirely. Randy took me into his group, and although I wish I could say it's been smooth sailing every step of the way since then, I can say that Randy has overflowed with support at every turn. I would be neither the mathematician nor the person I am today without Randy's continued help and guidance.

I would also like to thank my committee advisors and research collaborators, Rick Brown of WPI and Pat Bidigare of Synoptic Engineering. Not only have they provided the data for the electromagnetics portion of this project, but they have taken the time every week to discuss the research and offer guidance and ideas. The time spent talking research with them has been one of the highlights of my PhD career due to their continued kindness and insight.

I would like to thank my other two committee members, Frank Zou and Oren Mangoubi, for graciously donating their time for the proposal and defense of this dissertation. It was a pleasure to have their input and guidance in preparing this work.

I must also acknowledge and thank Luca Capogna, whose kindness and generosity guided me through the early years of my PhD and my formal exams. Luca helped me to become a mature mathematician and see the world of academic mathematics, and I will always be grateful for everything he's done for me.

I would like to thank everyone in the math program for their kindness, generosity, and friendship. Most of all, I must acknowledge Elisa Negrini, without whose friendship and encouragement I would have left the PhD program, and Ri-

uji Sato, a wonderful collaborator and a dear friend. I must also expressly thank Mia Barger for all her extraordinary work on our signal processing project; none of this would have been achieved without her work over the last year and a half.

Last, but most of all, I'd like to thank my family – my parents, Craig and Heidi, and my sisters, Veronica and Merina – for their continued love and support throughout this very long journey. I'd have never gotten here without them.

Papers Contributing to this Dissertation

Published First Author Papers

E. Witz, M. Barger, and R. Paffenroth, "Deep Learning for Range Localization via Over-Water Electromagnetic Signals," 2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA), 2021, pp. 1537-1544.

Extended Abstracts Submitted

E. Witz, M. Barger, and R. Paffenroth, "A Detailed Accounting of Noise in Neural Networks for Electromagnetic Signal Processing," Submitted to Asilomar Conference on Signals, Systems, and Computers, 2022

M. Barger, **E. Witz**, and R. Paffenroth, "Neural Network Output as Kalman Filter Input with Applications to Electromagnetics, Submitted to Asilomar Conference on Signals, Systems, and Computers 2022

Working Papers

E. Witz and R. Paffenroth, "Kronecker Networks and Piecewise Polynomial Activation," For submission to Frontiers in Applied Mathematics and Statistics.

Funding Proposals Based Upon Results in this Dissertation

DARPA - ”Messina: Statistical Modeling of AI Uncertainty with Applications to Passive Overwater Emitter Localization,” submitted by Synoptic Engineering to DARPA, June 2022

Funding and Technical Acknowledgements

Most of the results of this dissertation were obtained using a high-performance computing system, the Turing Research Cluster, provided to WPI through the NSF grant DMS-1337943.

The code which generates the predictions of the Maximum Likelihood Estimator was generously provided by D. Richard Brown of WPI.

Executive Summary

Overview of Our Work

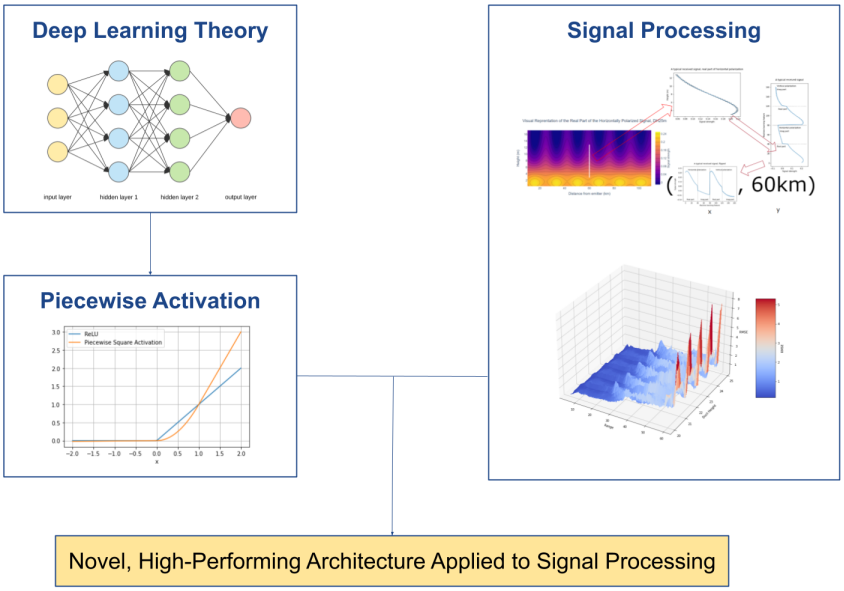

Figure 1: Overview of Our Main Work

Our focus in this work lies in the intersection of mathematics and data science, with applications in the physical sciences. For our theoretical work, we formulate a never-before-seen network architecture that allows us to peer into the heart of a neural network – we can see by our construction exactly how the network arrives at prediction and even glean insights about the data set itself from the behavior of the network. For our applied work, we make advancements in the signal processing space and develop a new framework for making predictions on signal problems by taking advantage of complex atmospheric conditions.

In Section 2, we begin the construction of our novel architecture by considering polynomial activation functions, namely, the activation function $\sigma(x) = x^2$. We demonstrate that these functions have remarkable algebraic properties, and

Table 1: Error rate for two models trained and tested on the MNIST data set of handwritten digits. Model 1 uses a standard LeakyReLU activation, while Model 2 uses the novel Piecewise Square activation.

Model	Description	Error Rate (%)
Model 1	An MLP with three hidden linear layers and LeakyReLU activation function, trained with negative log likelihood loss.	$2.28 \pm 0.17\%$
Model 2	An MLP with three hidden linear layers and Piecewise Square activation function, trained with negative log likelihood loss.	$2.21 \pm 0.00\%$

through the algebraic deconstruction of the network we develop in this work, we provide a path toward proving a Universal Approximation theorem for these functions. In Section 3, we develop a piecewise polynomial version of this square activation function with improved performance and stability properties. We demonstrate this by performing practical experiments on the MNIST data set.

Following this, we turn our attention toward applications in electromagnetics. Our major investigation here is to determine the distance between a signal transmitter and a receiver using only the received signal as input data. We conduct a series of investigations to determine the correct choices of input data, network architecture, and data transforms and preprocessing the optimize network performance on this problem. Additionally, we are able to compare the performance of this model against the optimal model in this domain, and Maximum Likelihood Estimator, and find that our performance is comparable to even the optimal model.

Finally, in Section 6, we combine the two realms of this work and evaluate our novel architecture of piecewise square activation on electromagnetic data. We find that not only is its performance comparable to standard architectures and the MLE, but it also opens a world of analysis inaccessible to other architectures.

Our Contributions

- We develop a theory for the construction of activation functions which provide us with both a rich algebraic theory as well as a deep level of interpretability from a mathematical perspective of machine learning. We develop a theoretical framework of polynomial activation functions that allow us to algebraically decompose a neural network function into its constituent parts, that is, to completely separate algebraically the network parameters from the input data.

- Using this decomposition, we provide a method by which a Universal Approximation Theorem can be proven for neural network with a polynomial activation function.

- We develop a novel activation function, the Piecewise Square Activation, and show that it naturally leads us to deeply analyze a neural network to give insight that other activations would not. We show that its algebraic structure allows one to easily compute local gradients of the network and demonstrate how this exposes the local geometry of the data set.

- We demonstrate how, through a discerning use of data selection and data transformation, one can build a robust model for deep learning in the signal processing domain. We show in a number of relevant cases that our models are able to achieve performance comparable to the optimal estimator in this setting, the Maximum Likelihood Estimator. We also demonstrate that these methods are applicable in a variety of simulated atmospheric conditions.

Road Map

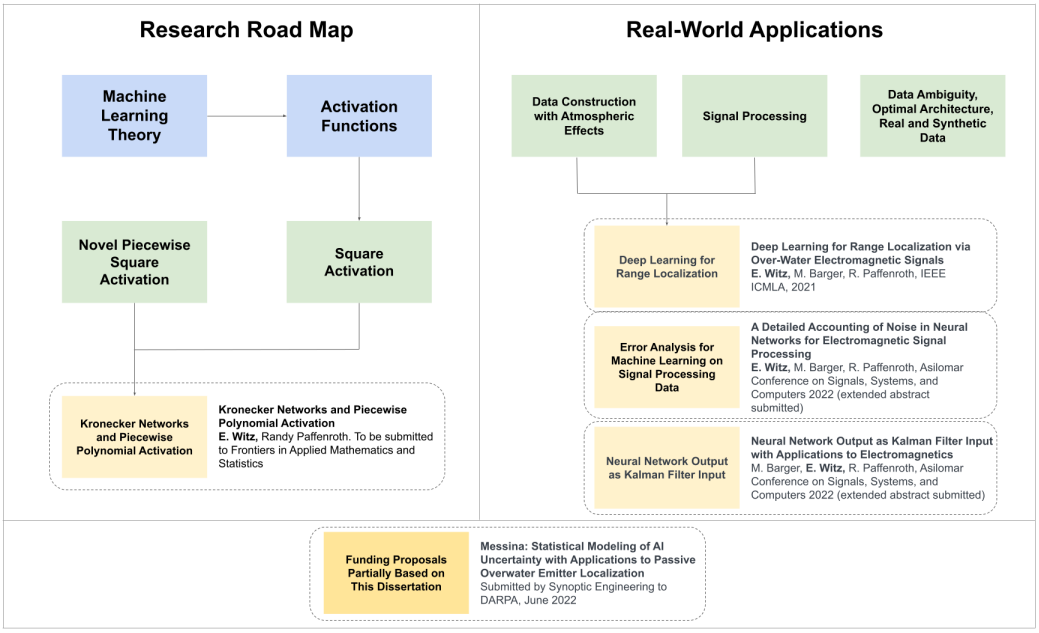

Figure 2: Research Road Map and Real-World Applications. This figure is a summary of our work over the last three years. In particular, the left-hand side details our theoretical work – the blue boxes are general areas of research which have inspired the present work, and the green boxes are our particular focus points. The right-hand side details our work in data-driven, real-world problems in signal processing and electromagnetics. Additionally, the four yellow boxes display our publications related to this work. At this time, we have one published first-author paper, one first-author paper in submission, one second-author paper in submission, and one first-author working paper. Finally, the orange box lists a funding proposal partially based on results in this dissertation.

Contents

1 Introduction

1.1 Background and Overview

While inhabiting the world of deep learning, one is very often split between two worlds – that of the beautiful theory and that of the useful application. Practitioners can point to the remarkable success deep learning methods have had in the fields of image recognition, language processing, the physical sciences, and many other spaces. Mathematicians can look at the remarkable set of density theorems, backed up by empirical evidence.

A long-held belief in the field of deep learning is that neural network methods are black boxes [9, 12, 38], that is, they are powerful objects that one can use to accomplish incredibly complex tasks, but studying them in a theoretical sense or determining their properties is enormously difficult. One can use a neural network to tell an image of a cat from an image of a dog, but it is often difficult or impossible to tell *how* this is being done.

The approach to studying deep learning we take in this work is to ask, "What is the simplest object which could be called a neural network? Are there any insights which could be gained from studying such objects?" Here, we offer answers to these questions in the form of so-called Kronecker neural networks, that is, neural networks whose activation functions are quadratic polynomials. We use this theoretical basis to develop a novel activation function and novel architectures.

In Section 2.2, we discuss the theoretical basis for our novel architecture, and that is the squared activation function. We begin by examining this activation function from a theoretical perspective. We are able to show algebraically in Theorem 2.2 that a neural network with a squared activation admits a decomposition into a

matrix product of a large matrix whose components are functions of the network parameters and a vector whose components are functions of the input data. This reformulation of the neural network object is remarkable since it takes a generally intractable, black-box object in a neural network and reveals its precise algebraic properties. In Section 2.3, we follow-up by considering the illuminative example of the one-dimensional case. In this case, we are able to make very precise algebraic statements about which functions are approximable by the network. The key in this section is that it reveals the structure of the accessible set of functions of the network, that is, the network has access to a submanifold of positive co-dimension of the set of polynomials. We find that, predictably, as the width of the network is increased, the dimension of the submanifold of accessible function increases until eventually all polynomials are accessible. The remarkable aspect of this construction is that it provides a path to proving a Universal Approximation Theorem for polynomial networks.

We substantiate these algebraic results empirically by showing the precise behavior of a small polynomial network under a standard training scheme. Additionally, these experiments reveal that the network formulated in this way with this activation function draws natural similarities to linear regression. The network has access to polynomial functions just as linear regression does, and it attempts to fit those functions to the input data in an optimal way. Since the optimal way of fitting polynomials to data is given by linear regression, we demonstrate that the network attempts to replicate this to the extent that it is able through the process of training by gradient descent.

Next, in Section 3, we construct a new activation function based on these ideas. With a pure polynomial activation network, one can run into issues of stability

and performance when one applies the network to real data. We address this and construct a new piecewise polynomial activation based on Rectified Linear Units (ReLU) which addresses these aspects of the activation. We demonstrate that this activation can be deconstructed in a similar way to the polynomial activation, leading to analyzable algebraic properties. Additionally, in Section 3.2, we give empirical evidence of the usefulness of such an approach to neural network formulation by testing the novel architecture on real data. We test this new construction on the MNIST data set and demonstrate that not only does it perform comparably to more standard activation functions, but we are also able to explicitly write down the function discovered by the network during the training process.

Beginning in Section 4, we discuss the major applied portion of the work. Our focus here is in the area of electromagnetics, particularly in range localization for over-water electromagnetic signals. For a signal transmitted over the ocean, we wish to determine the distance to the transmitter source using *only* the received signal along with a neural network model. We develop a particular data preprocessing scheme which is particularly effective for transforming the data into a form which is most efficiently learnable by the neural network methods.

In this over-water setting, atmospheric effects play a key role in the propagation of an electromagnetic signal. We demonstrate that despite the outsized effect evaporation ducts have on the received signal, a neural network with access to information about the evaporation duct as well as the electromagnetic signal is outperformed by a network which is given the electromagnetic signal alone. Additionally, we demonstrate that the reason for this is the low quality of atmospheric data available in practice.

We are also able in Section 5 to compare the results of our neural network mod-

els to a benchmark model, the Maximum Likelihood Estimator. Crucially, this model is the optimal solution for this problem, so it is important for the network to be able to perform well against it. We show that in many relevant cases, the network models are able to achieve comparable performance to the MLE model. Additionally, the network outperforms in MLE in terms of evaluation time in the case of complex atmospheric conditions.

Finally, in Section 6, we demonstrate that the theoretical framework developed in Sections 2 and 3 is useful for even the very complex data we encounter in the signal processing portion of the work. We demonstrate that as in the case of MNIST, this new activation is able to achieve results that are comparable to activation functions that are staples of the field such as LeakyReLU. Using the tools provided to us by the piecewise square activation function, we are also able to determine the precise structure of the function the network finds during the training process. We find that we are able to go in-depth into the precise way in which the network interacts with its input data in a way that data scientists typically cannot; in fact, we are able to exactly determine the way in which the network locally distinguishes nearby points. This comes from the fact that local partial derivatives and local gradients becomes very easy to compute when one uses an activation function that leads to a piecewise polynomial structure in the resulting function.

1.2 Related Work

The study of activation functions is as old as the formulation of neural networks themselves; the theoretical work goes back to the 1980s. The first major work in this domain is the original Universal Approximation Theorem by Cybenko [11], showing that networks constructed as sums of linear transforms with sigmoidal

activations applied are universal approximators. Universal approximation theorems tend to be very important in the deep learning space [10, 24, 27, 51, 57], since they give a "minimum viability" to a particular network construction. An interesting note is that in Cybenko's work, a very broad class of functions satisfy the sigmoidal property; however, polynomials are specifically excluded from this theorem. Regardless, part of this work serves to specifically study polynomial activation functions and demonstrate properties that make progress toward a universal approximation theorem for polynomial networks.

The most influential development in the realm of activation functions for the purpose of this work is the formulation of the Rectified Linear Unit (ReLU) activation [25, 36]. There are many ReLU-type activations, a few of which we list here. Softplus [18] is a C^∞ exponential approximation of ReLU. LeakyReLU [34] augments ReLU through the use of a gradient that does not vanish for negative inputs. Gaussian Error Linear Unit (GELU) [20], introduced in 2016, is a C^∞ version of ReLU that makes use of the CDF of a standard normal distribution. There is also the Scaled Exponential Linear Unit (SELU) [29], which allows the network to self-normalize leading to improved prediction. There are many other activation functions of ReLU-type and others, a survey of which is given in [3].

The way that activation functions are typically presented is to offer either a performance increase or a smoothness property that allows for improved training. In this work, we rather emphasize an improved analyzability property for networks that make use of our piecewise polynomial activation function. Polynomial or piecewise polynomial activation functions are not a novel construction [7, 28, 41, 52, 61], though very few are of ReLU-type. [28], for instance, offers a C^1 piecewise polynomial activation that are constant when $x < 0$ and $x > c$ for some

c, leading to a function which is more sigmoidal in overall shape. Our activation, by contrast, is a piecewise polynomial which goes to $\pm\infty$ as $|x| \to \infty$, giving much more of a ReLU (or more specifically, LeakyReLU) flavor. We demonstrate in this work that a function constructed in this way has nearly identical performance properties to LeakyReLU, while retaining the analyzability properties of networks with polynomial activations.

Analyzability or explainability is not a new concern in the realm of neural networks [31]. Being such complex objects performing potentially very important tasks, practitioners and researchers often find themselves wanting to explain how a neural network makes its predictions. [56] focuses on explainability through the use of architecture constraints, while [23, 53, 58] focus on explaining the predictions of graph neural networks. In the current work, we focus on explainability by way of studying the structure of the function discovered by the network. We analyze its local properties and relate this to the underlying data set. Surveys and reviews of work in explainable machine learning are given in [2, 8, 14, 55].

2 Kronecker Networks and Square Activation

2.1 Neural Networks, an Introduction

Neural networks are complex objects that come in all shapes, sizes, and varieties, but they all have a few things in common. We give a basic construction and a definition here.

The most important ingredient of a neural network is function composition. The idea is that even if one starts with a very simple function $f(x)$, by composing many simple functions $f_d(f_{d-1}(\dots f_2(f_1(x))))$, one can express very complicated functions of one's input x. We refer to these functions as layers.

The simplest possible functions, and the ones that are very often used as the building blocks of a neural network, are affine transformations. For an input vector $x \in \mathbb{R}^n$, these take the form

$$\mathbf{W}x + b,$$

where $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Neural networks work by composing these affine transformations using more $\mathbf{W}$s and bs:

$$\mathbf{W}_d(\dots \mathbf{W}_2(\mathbf{W}_1 x + b_1) + b_2) + \dots) + b_d.$$

However, this on its own is insufficient. A simple rearrangement by distribution shows that a composition of affine functions is still affine:

$$\mathbf{W}_d(\dots \mathbf{W}_2(\mathbf{W}_1 x + b_1) + b_2 + \dots) + b_d = \left(\prod_{i=d}^{1} \mathbf{W}_i\right) x + \sum_{i=1}^{d} \left(\prod_{j=d}^{i+1} \mathbf{W}_j\right) b_i,$$

where Π denotes a matrix product. Note that the order of the product matters as

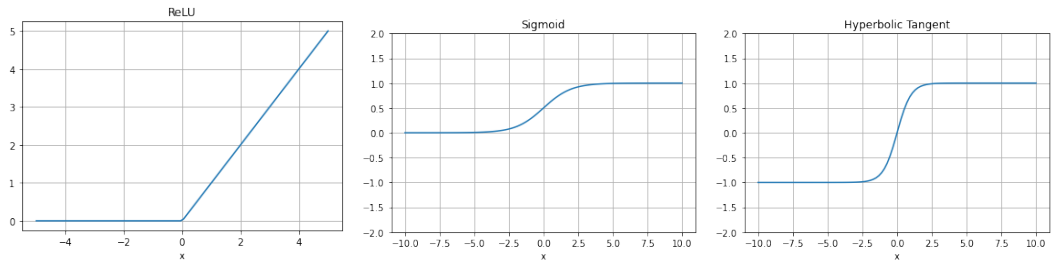

Figure 3: Sample activation functions ReLU, Sigmoid, and Hyperbolic Tangent tanh. Each activation function has different properties, though choosing the "correct" activation for a given problem can be difficult.

these are matrix products; the outermost layers appear first in the multiplication.

To remedy this and allow us to construct a full-fledged neural network, the final ingredient is the activation function σ. The activation function is what provides nonlinearity to each layer and allows the overall network to model nonlinear functions. Activation functions are typically defined as functions of one variable, and are applied componentwise to each layer of the network. There are many candidate activation functions one may use in a network, such as ReLU, Sigmoid [11], or Hyperbolic Tangent, shown in Figure 3.

Finally, if we collect the $\mathbf{W}$s and bs into a single notational parameter θ, all these ingredients are combined into a single neural network, which we will denote by f:

$$f(\theta, x) = \mathbf{W}_d\sigma(\dots\sigma(\mathbf{W}_2\sigma(\mathbf{W}_1x + b_1) + b_2) + \dots) + b_d. \tag{1}$$

Functions of this form are remarkably powerful, and through a discerning training process to select the parameters θ, they allow us to fit extremely complex functions of the input data x. Neural networks are typically visualized in images such as the one in Figure 4; in images such as this, the nodes represent elements of the vector $\sigma(\mathbf{W}x + b)$, while the arrows represent the action of each layer as it relates to the elements of the next layer.

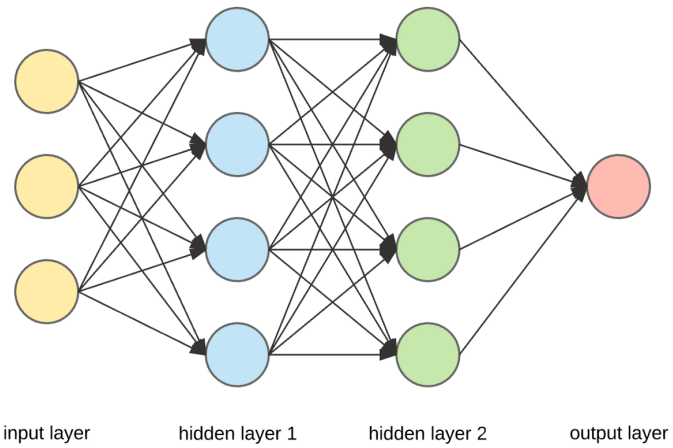

Figure 4: Visualization of a neural network. Image sourced from https://towardsdatascience.com/.

For problems in real applications, we typically start with a collection of data $\{(x_i, y_i)\}_{i=1}^N$, where N is the size of the data set, and we want to find a function $f(\theta, x)$ which best approximates y from x by making smart choices of θ. To that end, we define a Loss function to measure how *poorly* our network is fitting the data. A typical loss function is L2 loss, also known as Mean Squared Error (MSE), given by

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N} \|y_i - f(\theta, x_i)\|_2^2.$$

Finally, to train the network, we minimize the loss function by performing a stochastic variation of gradient descent, the non-stochastic version of which is given below:

$$\theta_{new} = \theta - \gamma \nabla_\theta \text{Loss},$$

where γ is a chosen parameter known as the learning rate. In practice, this is a simplification of what is generally a complex optimization process to find the best possible $\mathbf{W}$s and bs; more details can be found in [1, 5] or other textbooks.

Here, we will begin our analysis with the relatively uncommon activation func-

tion $\sigma(x) = x^2$. The key insight here is that when using the squared activation function $\sigma(x) = x^2$, we can use properties of Hadamard and Kronecker products to expand composed functions. Networks of this kind were considered by Berthiaume Paffenroth [7], though we in this work go deeper into the algebraic deconstruction and its implications.

2.2 Square Activation Function

Consider the network with square activation.

Assume we have $\hat{\mathbf{W}} \in \mathbb{R}^{m \times n}$, $\hat{x} \in \mathbb{R}^{n \times 1}$, and $\hat{b} \in \mathbb{R}^{m \times 1}$. Then, one can write the affine transformation

$$\hat{y} = \hat{\mathbf{W}}\hat{x} + \hat{b},$$

with $\hat{y} \in \mathbb{R}^{m \times 1}$. For notational convenience, since we will be taking products of sums in later steps, we wish to collect the $\hat{\mathbf{W}}$ and $\hat{b}$ into a single term. Thus, we rewrite this expression as the following:

$$\begin{bmatrix} \hat{y} \\ 1 \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{W}} & b \\ 0 \cdots 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{x} \\ 1 \end{bmatrix}.$$

So, defining

$$x = \begin{bmatrix} \hat{x} \\ 1 \end{bmatrix}, y = \begin{bmatrix} \hat{y} \\ 1 \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \hat{\mathbf{W}} & b \\ 0 \cdots 0 & 1 \end{bmatrix},$$

we arrive at the simplified form

$$y = \mathbf{W}x$$

to express each layer of a neural network.

We want to be able to apply the squared activation function here, i.e., we would

like to be able to write $(\mathbf{W}x)^2$. However, we have to be careful about what we mean when we write this, since $\mathbf{W}x$ is a vector and cannot be squared directly. Keeping in mind that we are applying an activation function, we note that we want to apply the square component-wise. Since a square is equivalent to self-multiplication, we may express this square as a component-wise product (known as the *Hadamard product*) of $\mathbf{W}x$ with itself. We express this with the notation

$$\mathbf{W}x \odot \mathbf{W}x.$$

Let us give a formal definition of a few matrix products here, since we will need them going forward.

Definition 2.1. For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, the **Hadamard Product** $\odot$ [32] is given by the component-wise product of $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1n}b_{1n} \\ a_{21}b_{21} & \ddots & & a_{2n}b_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1}b_{m1} & \cdots & & a_{mn}b_{mn} \end{bmatrix},$$

Definition 2.2. For two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$, the **Kronecker Product** $\otimes$[32] is

given by the following product of $\mathbf{A}$ and $\mathbf{B}$:

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & \cdots & a_{11}b_{1q} & \cdots & \cdots & a_{1n}b_{11} & a_{1n}b_{12} & \cdots & a_{1n}b_{1q} \\ a_{11}b_{21} & a_{11}b_{22} & \cdots & a_{11}b_{2q} & \cdots & \cdots & a_{1n}b_{21} & a_{1n}b_{22} & \cdots & a_{1n}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{11}b_{p1} & a_{11}b_{p2} & \cdots & a_{11}b_{pq} & \cdots & \cdots & a_{1n}b_{p1} & a_{1n}b_{p2} & \cdots & a_{1n}b_{pq} \\ \vdots & \vdots & & \vdots & \ddots & & \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots & & \ddots & \vdots & \vdots & & \vdots \\ a_{m1}b_{11} & a_{m1}b_{12} & \cdots & a_{m1}b_{1q} & \cdots & \cdots & a_{mn}b_{11} & a_{mn}b_{12} & \cdots & a_{mn}b_{1q} \\ a_{m1}b_{21} & a_{m1}b_{22} & \cdots & a_{m1}b_{2q} & \cdots & \cdots & a_{mn}b_{21} & a_{mn}b_{22} & \cdots & a_{mn}b_{2q} \\ \vdots & \vdots & \ddots & \vdots & & & \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{p1} & a_{m1}b_{p2} & \cdots & a_{m1}b_{pq} & \cdots & \cdots & a_{mn}b_{p1} & a_{mn}b_{p2} & \cdots & a_{mn}b_{pq} \end{bmatrix},$$

To take the Kronecker product of a matrix $\mathbf{A}$ with itself n times, we adopt the notation

$$\bigotimes_{i=1}^{1} \mathbf{A}.$$

Finally, we give one more definition, originally found in [26].

Definition 2.3. For two matrices $\mathbf{A} \in \mathbb{R}^{m \times n_1}, \mathbf{B} \in \mathbb{R}^{m \times n_2}$, the **Row-wise Khatri-Rao Product** $\bullet$ is given by the Kronecker product of the rows of $\mathbf{A}$ and $\mathbf{B}$, that is, if $\mathbf{A}$ and $\mathbf{B}$ are written as

$$\mathbf{A} = \begin{bmatrix} A_1 \\ \hline A_2 \\ \hline \vdots \\ \hline A_m \end{bmatrix}, \mathbf{B} = \begin{bmatrix} B_1 \\ \hline B_2 \\ \hline \vdots \\ \hline B_m \end{bmatrix},$$

where A_i, B_i denote the rows of the matrices $\mathbf{A}$ and $\mathbf{B}$, then the Khatri-Rao product $\mathbf{A} \bullet \mathbf{B} \in \mathbb{R}^{m \times n_1 n_2}$ is given by

$$\mathbf{A} \bullet \mathbf{B} = \begin{bmatrix} A_1 \otimes B_1 \\ \hline A_2 \otimes B_2 \\ \hline \vdots \\ \hline A_m \otimes B_m \end{bmatrix}.$$

We list a few important identities relating these matrix products:

Theorem 2.1. ***Properties of Kronecker, Hadamard, and Khatri-Rao Products****[48]* *Let* $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}, x, y \in \mathbb{R}^{n \times 1}$.

1. $(\mathbf{A}x) \odot (\mathbf{B}y) = (\mathbf{A} \bullet \mathbf{B})(x \otimes y)$
2. $(\mathbf{A}x) \otimes (\mathbf{B}y) = (\mathbf{A} \otimes \mathbf{B})(x \otimes y)$
3. $(\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$

Returning to the matter at hand, we apply the activation function $\sigma(x) = x^2$ component-wise as usual to the vector $\mathbf{W}x$. Then $y = (\mathbf{W}x)^2$ becomes the Hadamard product $y = \mathbf{W}x \odot \mathbf{W}x$. We may apply Theorem 2.1 and see that this is equivalent to $y = (\mathbf{W} \bullet \mathbf{W})(x \otimes x)$, where $\bullet$ is the row-wise Khatri-Rao product and $\otimes$ is the Kronecker product.

We may iterate this mapping by adding more layers. Consider

$$y = \mathbf{W}_3(\mathbf{W}_2(\mathbf{W}_1 x)^2)^2,$$

which is a network of two hidden layers and squared activation. We may perform the same algebraic analysis on this mapping:

$$\begin{aligned} y &= \mathbf{W}_3(\mathbf{W}_2(\mathbf{W}_1 x)^2)^2 \\ &= \mathbf{W}_3(\mathbf{W}_2((\mathbf{W}_1 x) \odot (\mathbf{W}_1 x)))^2 \\ &= \mathbf{W}_3((\mathbf{W}_2((\mathbf{W}_1 x) \odot (\mathbf{W}_1 x))) \odot (\mathbf{W}_2((\mathbf{W}_1 x) \odot (\mathbf{W}_1 x)))). \end{aligned}$$

We have that $(\mathbf{W}_1 x) \odot (\mathbf{W}_1 x))$ is a vector, to which the matrix $\mathbf{W}_2$ is being multiplied. Therefore, we may apply the mixed product identity once more:

$$\begin{aligned} y &= \mathbf{W}_3(\mathbf{W}_2 \bullet \mathbf{W}_2)(((\mathbf{W}_1 x) \odot (\mathbf{W}_1 x)) \otimes ((\mathbf{W}_1 x) \odot (\mathbf{W}_1 x))) \\ &= \mathbf{W}_3(\mathbf{W}_2 \bullet \mathbf{W}_2)(((\mathbf{W}_1 \bullet \mathbf{W}_1)(x \otimes x)) \otimes ((\mathbf{W}_1 \bullet \mathbf{W}_1)(x \otimes x))) \\ &= \mathbf{W}_3(\mathbf{W}_2 \bullet \mathbf{W}_2)(((\mathbf{W}_1 \bullet \mathbf{W}_1) \otimes (\mathbf{W}_1 \bullet \mathbf{W}_1))(x \otimes x \otimes x \otimes x). \end{aligned}$$

We find that, finally,

$$y = \mathbf{W}_3(\mathbf{W}_2 \bullet \mathbf{W}_2)\ ((\mathbf{W}_1 \bullet \mathbf{W}_1) \otimes (\mathbf{W}_1 \bullet \mathbf{W}_1))\ (x \otimes x \otimes x \otimes x) \tag{2}$$

By this construction, we have already proven the major theorem of the theoretical portion of this work. The case of a deeper network follows identically, with the primary difficulty being that of notation.

Theorem 2.2. *A neural network $f(\theta, x)$ of depth d, where θ represents the set of network parameters and x represents the input data, with the activation $\sigma(x) = x^2$, admits a decomposition into the form*

$$f(\theta, x) = P(\theta)\Phi(x).$$

Moreover, this decomposition is given by

$$f(\theta, x) = \left(\mathbf{W}_d \prod_{i=d-1}^{1} \bigotimes_{j=d-2}^{1} (\mathbf{W}_i \bullet \mathbf{W}_i)\right)\left(\bigotimes_{j=d-1}^{1} x\right)$$

Proof. See above. □

A decomposition of this form calls to mind theorems such as the Fisher-Neyman factorization theorem [17], which gives a characterization of a sufficient statistic with the existence of a decomposition of a probability density function $f_\theta(x)$ into functions $h(x)g_\theta(T(x))$, where T is a statistic. The difference here is that the Fisher-Neyman theorem tells us something about T, while the decomposition here tells us something about the relationship between f and θ, namely, it gives us an exact mapping from network parameters to polynomial coefficients.

2.3 The One-Dimensional Case

In reducing this network down to a matrix-vector operation, we have revealed the beautiful structure obscured by the function-compositional formulation of the network. Consider the final multiplicative term of this product, $x \otimes x \otimes x \otimes x$. Let us assume that the network has one-dimensional input. Then the x vector has the form

$$x = \begin{bmatrix} \hat{x} & 1 \end{bmatrix}^T$$

as stated earlier in the section, where ^{T} denotes the vector transpose. Let us consider Kronecker products of x with itself.

$$x \otimes x = \begin{bmatrix} \hat{x}^2 & \hat{x} & \hat{x} & 1 \end{bmatrix}^T$$

$$x \otimes x \otimes x = \begin{bmatrix} \hat{x}^3 & \hat{x}^2 & \hat{x}^2 & \hat{x} & \hat{x}^2 & \hat{x} & \hat{x} & 1 \end{bmatrix}^T$$

We see that as one takes Kronecker products of the x vector with itself up to n

times, all terms of the nth-order polynomial of $\hat{x}$ appear as the entries of the new vector. Finally, let us consider the structure of this vector when the input to the network is 2-dimensional. In this case,

$$x = \begin{bmatrix} x_1 & x_2 & 1 \end{bmatrix}^T,$$

$$x \otimes x = \begin{bmatrix} x_1^2 & x_1x_2 & x_1 & x_1x_2 & x_2^2 & x_2 & x_1 & x_2 & 1 \end{bmatrix}^T.$$

Likewise, in this case, the terms of the multivariate polynomial of variables x_1, x_2 appear as entries in the vector $x \otimes x$.

Now, recall the object of interest in this discussion – we are examining the structure of a neural network with square activation function. In the one-dimensional case, we could simply expand the polynomial $y = w_3(w_2(w_1x + b_1)^2 + b_2)^2 + b_3$, write the polynomial term by term, and see the structure of the resulting function. However, deriving this structure using the matrices and Kronecker and Khatri-Rao products of Equation 2 is illuminative as it establishes familiarity with the objects. To that end, let us expand this formula piece by piece, starting with the Khatri-Rao product $\mathbf{W}_1 \bullet \mathbf{W}_1$:

$$\mathbf{W}_1 \bullet \mathbf{W}_1 = \begin{bmatrix} w_1 & b_1 \\ 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} w_1 & b_1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} w_1^2 & w_1b_1 & w_1b_1 & b_1^2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Next, we see this Khatri-Rao product self-multiplied into a Kronecker product:

$$
\begin{aligned}
&(\mathbf{W}_1 \bullet \mathbf{W}_1) \otimes (\mathbf{W}_1 \bullet \mathbf{W}_1) \\
&= \begin{bmatrix}
w_1^4 & w_1^3 b_1 & w_1^3 b_1 & w_1^2 b_1^2 & w_1^3 b_1 & w_1^2 b_1^2 & w_1^2 b_1^2 & w_1 b_1^3 & w_1^3 b_1 & w_1^2 b_1^2 & w_1^2 b_1^2 & w_1 b_1^3 & w_1^2 b_1^2 & w_1 b_1^3 & w_1 b_1^3 & b_1^4 \\
0 & 0 & 0 & w_1^2 & 0 & 0 & 0 & w_1 b_1 & 0 & 0 & 0 & w_1 b_1 & 0 & 0 & 0 & b_1^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & w_1^2 & w_1 b_1 & w_1 b_1 & b_1^2 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\end{aligned}
$$

We are ready to add the second layer, in the form of $\mathbf{W}_2 \bullet \mathbf{W}_2$:

$$
(\mathbf{W}_2 \bullet \mathbf{W}_2)(\mathbf{W}_1 \bullet \mathbf{W}_1) \otimes (\mathbf{W}_1 \bullet \mathbf{W}_1) = \begin{bmatrix} A & B & C & D \end{bmatrix},
$$

where

$$
A = \begin{bmatrix}
w_1^4 w_2^2 & w_1^3 w_2^2 b_1 & w_1^3 w_2^2 b_1 & w_1^2 w_2^2 b_1^2 + w_1^2 w_2 b_2 \\
0 & 0 & 0 & 0
\end{bmatrix},
$$

$$
B = \begin{bmatrix}
w_1^3 w_2^2 b_1 & w_1^2 w_2^2 b_1^2 & w_1^2 w_2^2 b_1^2 & w_1 w_2^2 b_1^3 + w_1 w_2 b_1 b_2 \\
0 & 0 & 0 & 0
\end{bmatrix},
$$

$$
C = \begin{bmatrix}
w_1^3 w_2^2 b_1 & w_1^2 w_2^2 b_1^2 & w_1^2 w_2^2 b_1^2 & w_1 w_2^2 b_1^3 + w_1 w_2 b_1 b_2 \\
0 & 0 & 0 & 0
\end{bmatrix},
$$

$$
D = \begin{bmatrix}
w_1^2 w_2^2 b_1^2 + w_1^2 w_2 b_2 & w_1 w_2^2 b_1^3 + w_1 w_2 b_1 b_2 & w_1 w_2^2 b_1^3 + w_1 w_2 b_1 b_2 & w_2^2 b_1^4 + 2 w_2 b_1^2 b_2 + b_2^2 \\
0 & 0 & 0 & 1
\end{bmatrix}.
$$

After performing the final matrix multiplication with $\mathbf{W}_3$, multiplying by the vector $x \otimes x \otimes x \otimes x$, and combining like terms, we arrive at the final structure of the polynomial network:

$$y = \begin{bmatrix} (w_1^4 w_2^2 w_3)x^4 + (4w_1^3 w_2^2 w_3 b_1)x^3 + (6w_1^2 w_2^2 w_3 b_1^2 + 2w_1^2 w_2 w_3 b_2)x^2 \\ \quad +(4w_1 w_2^2 w_3 b_1^3 + 4w_1 w_2 w_3 b_1 b_2)x + (w_2^2 w_3 b_1^4 + 2w_2 w_3 b_1^2 b_2 + w_3 b_2^2 + b_3) \\ 1 \end{bmatrix}$$

Now consider the structure of each term of the output vector y. Each entry of the y vector is exactly a fourth-order multivariate polynomial of the entries in the x vector. In this way, this architecture draws us a very natural comparison to the most basic prediction technique: linear regression. That is, the neural network exactly has access to some set of polynomials of degree 4. Given a data set of inputs and outputs of a function f, it is known that the optimal way to approximate f is given by the methods of linear regression. We explore this relationship further in the following sections.

2.3.1 What functions are in this family?

This section follows in principle part of the construction of spanning dimension of Berthiaume and Paffenroth [7], though we expand on the work by giving an explicit characterization of the function for which the gradient is computed as well as a more explicit characterization of the manifold of accessible functions.

As a starting point, let us assume our data is one-dimensional in both input and output. We may then write this in an explicit form in the following way. Note that in this setting, each $\mathbf{W}$ has the explicit form

$$\mathbf{W}_i = \begin{bmatrix} w_i & b_i \\ 0 & 1 \end{bmatrix}$$

where each w_i, b_i is a scalar.

Denote the neural network by $f_{\mathbf{W}}(x) = \mathbf{W}_3(\mathbf{W}_2(\mathbf{W}_1 x)^2)^2$. After performing the matrix multiplications as before, we arrive at the formula

$$\begin{aligned} f_{\mathbf{W}}(x) &= (w_1^4 w_2^2 w_3)x^4 + \\ &\quad (4w_1^3 w_2^2 w_3 b_1)x^3 + \\ &\quad (6w_1^2 w_2^2 w_3 b_1^2 + 2w_1^2 w_2 w_3 b_2)x^2 + \\ &\quad (4w_1 w_2^2 w_3 b_1^3 + 4w_1 w_2 w_3 b_1 b_2)x + \\ &\quad (w_2^2 w_3 b_1^4 + 2w_2 w_3 b_1^2 b_2 + w_3 b_2^2 + b_3) \end{aligned} \tag{3}$$

We have discussed the derivation of this formula in detail; at this point we would like to move on to the *meaning* of this formula. We see that as a function, this neural network is a fourth order polynomial of its scalar input x. By noting that we may choose the w's and b's, we can see that the network clearly has access to some subset of the 5-dimensional set of polynomials of degree 4. What is not clear, however, is exactly which polynomials are accessible by the network. We would like to find out exactly which ones these are. To this end, we define the *solvable set*.

Definition 2.4. Solvable Set The solvable set is the set of polynomials

$$p(x) = ax^4 + bx^3 + cx^2 + dx + e$$

for which there exist coefficients $w_1, w_2, \ldots, b_3$ such that $f_{\mathbf{W}}(x) = p(x)$.

Theorem 2.3. *The solvable set is the set of all $p(x) = ax^4 + bx^3 + cx^2 + dx + e$ such that one of the following is true:*

1. *All coefficients are 0.*

2. $a \neq 0$, $b \neq 0$, *and* $d = \frac{4abc - b^3}{8a^2}$.

Moreover, the choice of $w_1, w_2, \ldots, b_3$ *has two degrees of freedom.*

Proof. Denote our free parameters by s and t. Set

$$w_1 = \frac{b}{4a}, w_2 = \frac{32a^3 s}{b^3}, w_3 = \frac{b^2}{4a}\frac{1}{s^2},$$

$$b_1 = t, b_2 = \left(-\frac{32a^3 t^2}{b^3} + \frac{c}{b} - \frac{b}{4a} \right) s, b_3 = e - \left(\frac{c}{b} - \frac{b}{4a} \right) s.$$

□

Theorem 2.3 gives an exact characterization of which functions are explicitly accessible to the network, that is, for which polynomials with coefficients a, b, c, d, e one can find a set of network parameters $w_1, w_2, \ldots, b_3$ such that the coefficients of the resulting network are exactly equal to the coefficients of the original polynomial. However, knowing which functions are exactly solvable by our system of polynomials is insufficient when discussing neural networks, since we must also consider any polynomial which can be approximated arbitrarily well by the neural network. To that end, we are also interested in density results, not merely results on exact solutions.

2.4 Fittable Functions and the Solvable Set

We now investigate which polynomials of the form

$$p(x) = \hat{a}x^4 + \hat{b}x^3 + \hat{c}x^2 + \hat{d}x + \hat{e}$$

we can approximate using polynomials of the form

$$p(x) = ax^4 + bx^3 + cx^2 + \frac{4abc - b^3}{8a^2}x + e.$$

The set of polynomials of this form is exactly the solvable set that our neural network has access to. This leads to our first density theorem.

Proposition 2.1. The solvable set is dense in the set of parabolas $p(x) = \hat{c}x^2 + \hat{d}x + \hat{e}$.

Proof. It is clear that since b_3 can take on any value, any polynomial in the solvable set can take on any value in the constant term. Therefore, we consider only parabolas $\hat{c}x^2 + \hat{d}x$.

It suffices to show that there exist sequences a_n, b_n, c_n such that $a_n \to 0, b_n \to 0, c_n \to \hat{c}$, with the constraint that

$$\frac{4a_n b_n c_n - b_n^3}{8a_n^2} = \hat{d}.$$

Set $c_n = \hat{c}$. The solution to the polynomial in two variables

$$4a_n b_n \hat{c} - b_n^3 = 8a_n^2 \hat{d}$$

is given by

$$\begin{aligned} a_n &= \frac{\sqrt{b_n^2(\hat{c}^2 - 2\hat{d}b_n)} + \hat{c}b_n}{4\hat{d}} \\ &= O(b_n) \text{ as } b_n \to 0. \end{aligned}$$

Then a_n and b_n may simultaneously go to 0, and so the solvable set is dense in the set of parabolas. □

We now prove a similar proposition for quartic polynomials. In this case we are not able to approximate every polynomial, only those with 0 third and first order terms.

Proposition 2.2. The solvable set is dense in the set of even quartic polynomials $p(x) = \hat{a}x^4 + \hat{c}x^2 + \hat{e}$.

Proof. As before, we may ignore the constant term. In this case, it suffices to show that there exist sequences a_n, b_n, c_n such that $a_n \to \hat{a}, b_n \to 0$, and $c_n \to \hat{c}$ with the constraint that

$$\frac{4a_n b_n c_n - b_n^3}{8a_n^2} \to 0.$$

Set $a_n = \hat{a}$ and $c_n = \hat{c}$. As $b_n \to 0$, $\frac{4\hat{a}b_n\hat{c} - b_n^3}{8\hat{a}^2} \to 0$ as well. □

We now prove a negative density result. The network is unable to approximate any monomials of third degree.

Proposition 2.3. The solvable set is not dense in the set of polynomials of the form $p(x) = \hat{b}x^3$.

Proof. Let $b_n \to \hat{b}$. Since $b_n \to \hat{b}$, the only way for

$$\frac{4a_n b_n c_n - b_n^3}{8a_n^2} \to 0,$$

is for $|a_n| \to \infty$. So either the fourth order term goes to infinity or the first order term is bounded away from 0. In either case, polynomials in the solvable set are not dense in the set of monomials of third order. □

2.5 Higher Degree Polynomials

This line of reasoning can be extended to higher degree polynomials. Let us consider the case where the neural network has depth d and the ith layer has width m_i. Consider the mapping from neural network parameters to polynomial coefficients provided naturally by our construction, namely the mapping from $\mathbb{R}^{m_1 \times n} \times \cdots \times \mathbb{R}^{1 \times m_d} \to \mathbb{R}^{2^{d-1}}$ given by

$$(\mathbf{W}_1, \ldots, \mathbf{W}_d) \mapsto \mathbf{W}_d \prod_{i=d-1}^{1} \bigotimes_{j=d-2}^{1} (\mathbf{W}_i \bullet \mathbf{W}_i)$$

In the three-layer case, if additional width is added to the network, the solvable set of polynomials accessible to the network extends to the entire 5-dimensional space of fourth order polynomials. We conjecture that this behavior extends to the case of higher degree polynomials as well, that is, if m_1, m_2, ..., m_d are large enough, the solvable set is always dense in the space of polynomials of degree 2^{d-1}.

We expect this to be the case. It is clear that as $m_1, m_2, \ldots, m_d$, are increased, the dimension of the manifold of accessible polynomials must always either stay the same or increase, up to a maximum of $2^{d-1} + 1$. Unless there is an unseen mechanism by which the dimension stops increasing somewhere below this maximum dimension, the manifold of accessible polynomials would always eventually become dense in the space of all polynomials of degree 2^{d-1}.

If this were true, it would constitute a formal proof of an idea that lies in the folklore of deep learning – that networks with polynomial activation functions are universal approximators if one allows them arbitrary depth. Previous theorems, such as Cybenko's classical Universal Approximation Theorem[11], demonstrate that polynomials do not serve as a universal approximator when one only consid-

ers networks of a single layer.

3 Piecewise Square Activation

3.1 Definition and Basic Properties

We can see that for a squared activation network, there is beautiful theory. We can develop explicit formulae that beautifully tease apart the various aspects of the network. So why is this type of activation not present in the neural network literature? There are two principal reasons. The first is that theorems such as Cybenko's Universal Approximation Theorem [11], the characterization of functions that serve as universal approximators explicitly exclude polynomials. This is not the primary reason, however, as this difficulty can most likely be removed by considering a network with polynomial activation of sufficient depth and width and applying the Stone-Weierstrass Theorem [50]. In practice, the reason polynomial activation functions are not used is that of stability. It is very difficult to train a network with polynomial activation, and they tend to not perform well in practice. We will see an example of this later on when showing a performance comparison between several candidate activation functions in Section 3.2.

We would like to construct a new activation function which has the properties of ReLU, while giving the interpretive and theoretic power of a squared activation function. There are many similar functions to ReLU, each of which claims a particular advantage in terms of training or theoretical properties. For example, Leaky ReLU is an invertible version of ReLU. Being invertible, it preserves information in the information-theoretic sense and in practice avoids the issue of vanishing gradient.

We would like an activation function that makes use of the structure we already have while maintaining the high performance properties of ReLU-type activation

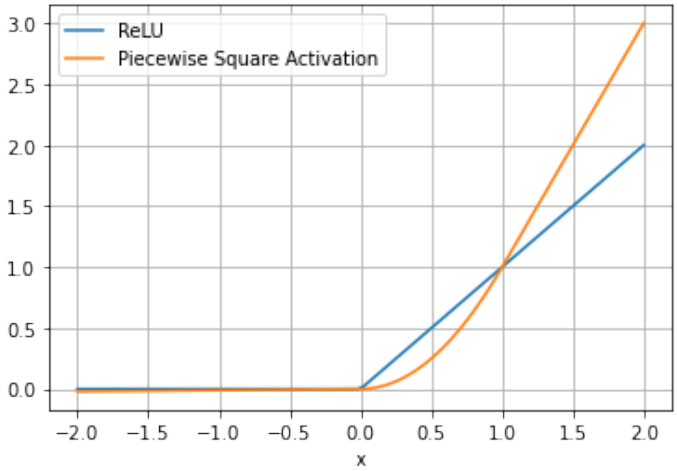

Figure 5: Graphical representation of a Piecewise Square Activation compared to the standard ReLU activation.

functions. To this end, we define a new activation function in the following way:

$$\text{Piecewise Square Activation} = \begin{cases} 0.01x - .000025 & x \leq .005 \\ x^2 & .005 \leq x \leq 1 \\ 2x - 1 & 1 \leq x \end{cases} \tag{4}$$

This function is C^1 and a piecewise polynomial, which gives us the algebraic properties we want out of an activation as well as the performance one would expect out of a ReLU-type activation function. We will see an example of the performance of this activation function later on.

This activation claims a particular advantage over other ReLU-type activations in the following way: we are able to explicitly enumerate the function discovered by the network during the training process through the use of its piecewise polynomial structure. Let us assume as before that we have an n-dimensional input space to our neural network, whose elements are denoted by $x \in \mathbb{R}^n$. Let us also return to the standard notation of referring to the parameters of the neural network

by $W_i \in \mathbb{R}^{m_i \times m_{i-1}}$ and $b_i \in \mathbb{R}^{m_i}$, where m_i is the size of the ith hidden layer.

We consider once more the action of the first layer onto the input vector x:

$$\begin{aligned} f_N(\theta, x) &= W_1 x + b_1 \\ &= \begin{bmatrix} w_{1,1,1} & \cdots & w_{1,1,n} \\ \vdots & \ddots & \vdots \\ w_{1,m_1,1} & \cdots & w_{1,m_1,n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,m_1} \end{bmatrix} \end{aligned} \tag{5}$$

When the multiplication and addition are performed and an activation function is applied, this may become, as an example,

$$\begin{aligned} f_N(\theta, x) &= \sigma(W_1 x + b_1) \\ &= \begin{bmatrix} .01(W_{1,1} \cdot x + b_{1,1}) - .000025 \\ (W_{1,2} \cdot x + b_{1,2})^2 \\ \vdots \\ 2(W_{1,m_1} \cdot x + b_{1,m_1}) - 1 \end{bmatrix}, \end{aligned} \tag{6}$$

where $W_{1,j}$ denotes the jth row of the matrix W_1.

As we have written it here, this layer is a function with n inputs and m_1 outputs. We wish to consider a slice of this function in a single direction, with all other variables constant. So let us consider the slice of $\mathbb{R}^n$ given by $(x_1, a_2, a_3, \ldots, a_n)$, where $a_1, \ldots, a_n$ are constant and x_1 remains variable. We consider what each layer looks like in this case:

$$\begin{aligned} f_N(\theta, x) &= W_1 x + b_1 \\ &= \begin{bmatrix} w_{1,1,1} & \cdots & w_{1,1,n} \\ \vdots & \ddots & \vdots \\ w_{1,m_1,1} & \cdots & w_{1,m_1,n} \end{bmatrix} \begin{bmatrix} x_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} b_{1,1} \\ \vdots \\ b_{1,m_1} \end{bmatrix} \end{aligned} \tag{7}$$

This expression is now a function of one-dimensional input and m_1 outputs, and with a piecewise square activation, we may explicitly write any particular element of the output of this function as a polynomial of x_1. We may also continue this process of applying $\sigma(W_i x + b_i)$ with every layer until finally reaching the final polynomial used to make predictions for a given point, which is what we do in the following.

Since m_1 may in theory be large, we may also do this using an algorithm. In code, we express the vector containing the x's and a's as a coefficient matrix of a first-order vector of polynomials denoted by P, i.e.,

$$\begin{bmatrix} x_1 \\ a_2 \\ \vdots \\ a_{784} \end{bmatrix} = \begin{bmatrix} 1x_1 + 0 \\ 0x_2 + a_2 \\ \vdots \\ 0x_{784} + a_{784} \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 \\ 0 & a_2 \\ \vdots & \vdots \\ 0 & a_{784} \end{bmatrix} = \begin{bmatrix} P_1 & P_2 \end{bmatrix} = P.$$

To generate the polynomial associated with a specific input index k and data point a, we simply construct the coefficient matrix P corresponding to a variable x_k and pass this P through the network while keeping track of which piece of the piecewise square activation function is being applied to every element. We add columns to P as necessary, as the power of the polynomials grow due to squares being applied.

For the example listed above in Equation 6, after one layer, this would become

$$
\begin{aligned}
& \sigma(W_1 P + b_1) \\
= \; & \sigma\left(\begin{bmatrix} W_{1,1} \cdot P_1 & W_{1,1} \cdot P_2 + b_{1,1} \\ W_{1,2} \cdot P_1 & W_{1,2} \cdot P_2 + b_{1,2} \\ \vdots & \vdots \\ W_{1,400} \cdot P_1 & W_{1,400} \cdot P_2 + b_{1,400} \end{bmatrix}\right) \\
= \; & \begin{bmatrix} 0 & .01 w_{1,1,1} & .01(W_{1,1} \cdot P_2 + b_{1,1}) - .000025 \\ w_{1,2,1}^2 & 2w_{1,2,1}(W_{1,2} \cdot P_2 + b_{1,2}) & (W_{1,2} \cdot P_2 + b_{1,2})^2 \\ \vdots & \vdots & \vdots \\ 0 & 2w_{1,1,400} & 2(W_{1,400} \cdot P_2 + b_{1,400}) - 1 \end{bmatrix}
\end{aligned}
\tag{8}
$$

Finally, we may generate polynomials associated with specific data points by repeating this process for every layer in the network. The complete algorithm for generating these polynomials is given in Algorithm 1.

3.2 Image Classification with Novel Activation

As an initial test of this activation function, we will focus in this portion on training and testing a model with the Piecewise Square activation function in the area of image classification. We will use the benchmark data set MNIST[30] provided by the PyTorch[42] Torchvision[35] library. We will demonstrate through these experiments that while unconventional in its formulation, the Piecewise Square activation performs comparably to more standard activation functions. We will use the activation function LeakyReLU as the benchmark activation to which we may compare our novel one.

We first devise a simple experiment in which we train two models, one with

Algorithm 1 An algorithm for generating the predictive polynomials for a given network and input data point. For convenience, rows and columns of W's, b's, and the data a have been transposed, since this is how these methods are typically implemented in code.

Input: $a \in \mathbb{R}^{1\times n}$, where a is a given input data point and n is its dimension
$W_1 \in \mathbb{R}^{m_1\times n}, \ldots, W_d \in \mathbb{R}^{m_d\times m_{d-1}}$, $b_1 \in \mathbb{R}^{m_1\times 1}, \ldots, b_d \in \mathbb{R}^{m_d\times n_d}$, where m_i is the size of the ith layer and $W_1, \ldots, W_d, b_1, \ldots, b_d$ are the parameters of a trained neural net.
k, where k is an integer between 0 and $n-1$ representing the index for which to compute the predictive polynomial.
Return: P, a matrix of polynomial coefficients representing slices of the function generated by the neural network.

```
P ∈ R^{2^{d-1}+1 × n} is a matrix of zeros.
P[0, k] ← 1
P[1, :] ← a
P[1, k] ← 0
L ← a                                          ▷ L tracks the output of each layer
for i = 1 ... d do
    P ← P W_1^T
    P[0, :] ← P[0, :] + b_1
    L ← PiecewiseSquare(L W_1^T + b_1)
    for j = 1 ... m_i do
        if L[i] < .000025 then
            P[:, j] ← .01 * P[:, j]
            P[0, j] ← P[0, :] − .000025
        end if
        if .000025 ≤ L[i] ≤ 1 then
            n ← number of nonzero entries of P[:, j]    ▷ # of coefficients currently
            P[0 : 2n − 1, j] ← np.convolve(P[0 : n, j], P[0 : n, j])
        end if
        if L[i] > 1 then
            P[:, j] ← 2 * P[:, j]
            P[0, j] ← P[0, :] − 1
        end if
    end for
end for
```

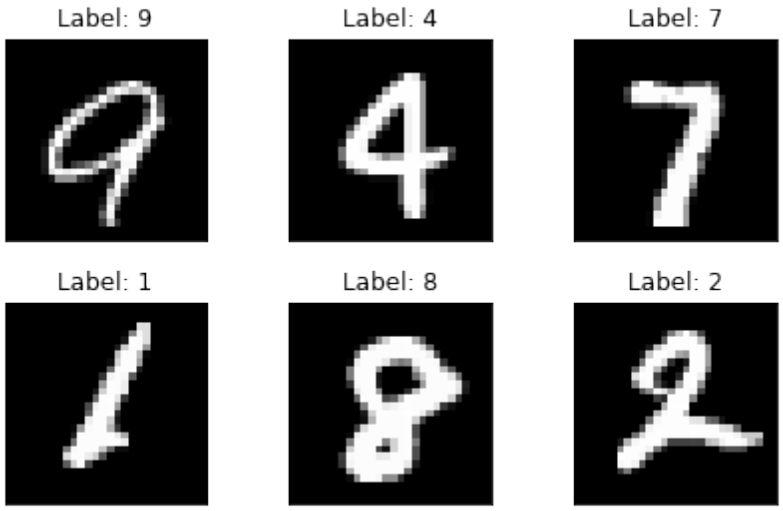

Figure 6: Select examples from the MNIST data set of handwritten digits. We will use this data set as a preliminary test to evaluate our novel activation function on a standard benchmark.

a LeakyReLU activation and one with a Piecewise Square activation, to classify digits on the handwritten digit data set MNIST.[1] Select examples of MNIST are shown in Figure 6. We will show that this new activation performs comparably to the standard one, while keeping in mind the additional algebraic structure provided by an activation of this type. These models are listed in more detail in Table 2.

For this experiment, Model 1 is a simple, fully connected MLP with three hidden layers. The LeakyReLU activation function is applied after the first three layers, and LogSoftmax is applied before the final output. Model 2 is intentionally constructed identically, save for the crucial difference in activation function. Model 2 uses the Piecewise Square activation after the first three layers instead of the LeakyReLU activation. Both models are trained for 10 epochs using SGD optimization from the "torch.optim" package, a batch size of 64, and learning rate

[1]Select code elements in this experiment sourced from https://nextjournal.com/gkoehler/pytorch-mnist.

Table 2: Error rate for two models trained and tested on the MNIST data set of handwritten digits. Model 1 uses a standard LeakyReLU activation, while Model 2 uses the novel Piecewise Square activation. The model architecture is identical otherwise.

Model	Description	Error Rate (%)
Model 1	An MLP with three hidden linear layers and LeakyReLU activation function, trained with negative log likelihood loss.	$2.33 \pm 0.11\%$
Model 2	An MLP with three hidden linear layers and Piecewise Square activation function, trained with negative log likelihood loss.	$2.08 \pm 0.20\%$

$\gamma = .01$.

While this difference in activation may seem small, it changes the structure of the function resulting from the neural network dramatically. As a continuous but not C^1 function with a corner, the function generated when using a LeakyReLU activation is continuous piecewise planar. It fits functions by tiling its input space with piecewise planes; this allows the function flexibility when it comes to fitting, but leads to functions that are exceptionally hard to parse and understand. In some sense, it could be considered a microcosm of deep learning as a whole. Piecewise polynomial activation, on the other hand, fits data with piecewise polynomials functions of its input. While these can still be difficult to write down and analyze for high-dimensional input, the task is much more tractable as one can track coefficients of the components of the input vector as they pass through the network as shown in the previous section. This allows us to actually write down the resulting function that the neural network finds through training using Algorithm 1.

After training models of the two types as specified earlier, we record the error rate of their predictions as the proportion of incorrectly classified images in the test

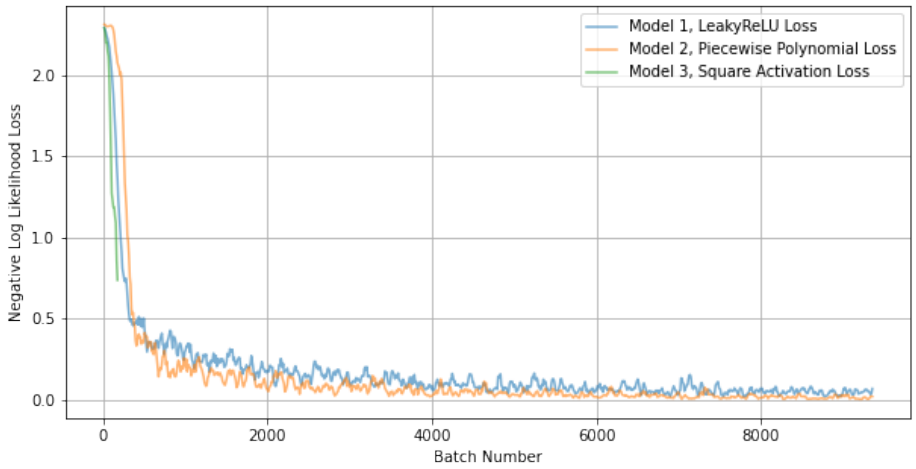

Figure 7: Comparison of the training losses of Model 1, which uses the LeakyReLU activation function, and Model 2, which uses the Piecewise Polynomial activation, when trained on the MNIST data set. Model 2 slightly outperforms Model 1 in terms of loss while training, though the difference is marginal and varies based on the choice of random seed – Models 1 and 2 can each outperform the other based on initialization. We have also plotted the performance of the square activation function; it fails due to instability after training for 20% of one epoch. For ease of reading, we have plotted a rolling average of the training loss.

data set. As we can see in Table 2, Model 2 outperforms Model 1 in this particular example, though the difference is small and depends on the random seed one chooses – either model can outperform the other depending on initialization. To generate the plus or minus values, we trained each model 5 times and recorded the max and min of the error rates of the 5 trials.

3.3 Model Comparison and Analysis

Figure 7 displays the training losses of the two networks as they train on MNIST. We display a rolling average of the training losses in Figure 7 to more easily compare the two loss curves. As we can see, Model 2 generally has the slightly lower loss throughout the training process, though Model 1 has a slightly lower classification error rate as shown in Table 2.

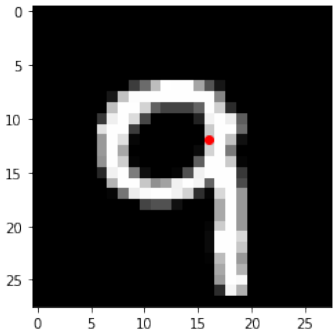

Figure 8: MNIST Example for which we may apply the polynomial generation algorithm, Algorithm 1. Pixel 352 is highlighted, which is the pixel we consider for this example.

While the performance of the two networks is similar, we will begin to demonstrate in this section the advantage of using an activation function like the Piecewise Square activation. By using Algorithm 1, we are able to determine the shape of the function found by the neural network.

For this illustration, we take an example of an image in MNIST, then choose a pixel (i.e., choose an element of the input vector x as in the previous section), then apply Algorithm 1 to view the polynomials found in the network in an area around that point. For this example, we choose the pixel highlighted in Figure 8, which corresponds to x_{352} if we consider the image an input vector.

There are two things to note about the polynomials generated for this example. First is that since we are making classification predictions with ten classes, each image and vector component (i.e. pixel) will have ten associated polynomials, since the output of the neural network is 10-dimensional. Second, note that in the architecture of our network, a Log Softmax is applied to each component the output of the final layer. For the purposes of polynomial generation, we display the function before the Log Softmax is applied.

The output of Algorithm 1 applied to the image and pixel in Figure 8 is given in

Table 3: Polynomials associated with each class prediction for the image and pixel in Figure 8. Polynomial terms with coefficients less than .001 are not shown for ease of reading.

Output Element	Polynomial
Predicting 0:	$-2.193 + 0.09x$
Predicting 1:	$-8.491 + 0.216x - 0.001x^2$
Predicting 2:	$-1.19 + 0.063x - 0.001x^2$
Predicting 3:	$2.627 - 0.082x$
Predicting 4:	$3.581 - 0.03x$
Predicting 5:	$1.314 - 0.234x + 0.004x^2$
Predicting 6:	$-7.789 + 0.032x$
Predicting 7:	$1.103 + 0.234x - 0.001x^2$
Predicting 8:	$2.314 - 0.223x - 0.001x^2$
Predicting 9:	$17.395 - 0.093x + 0.001x^2$

Table 3. We notice several things immediately. The first is that the polynomial associated with the prediction "9" has the largest constant term out of the ten options. This is a consistent trend among the images we have tested with this method; the network tends to put its predictions into the constant term in the polynomial rather than using particular elements of x.

We may even measure this empirically. We select several pixels in the image given in Figure 8 and compute the associated polynomials using Algorithm 1. We then add up the magnitudes of the coefficients associated with each order of the polynomials; these results are given in Figure 9. We see that the constant terms are an order of magnitude larger than the first order terms, the first order terms are order of magnitude larger than the second order terms, and so on.

Let us be clear about what exactly we are showing in Tables 3 and 4. If we look at the local structure of the function associated with the neural network, the

function is exactly a multivariate polynomial of its 784 inputs. The degree of the polynomial depends on precisely how often the function x^2 is applied as a data point passes through the network, but as a rough sketch, the function takes the form

$$f(x,\theta) = \sum_{i_0=0}^{8}\sum_{i_1=0}^{8}\cdots\sum_{i_{783}=0}^{8}\theta_{i_0 i_1 \ldots i_{783}} x_0^{i_0} x_1^{i_1}\ldots x_{783}^{i_{783}}, \tag{9}$$

where θ is a polynomial coefficient consisting of elements of the ws and bs. What we are showing in this table is a slice of that function with all variables except for one held constant, that is, using our notation from Section 3.1,

$$f(x,\theta) = \sum_{i_{352}=0}^{8}\left(\theta_{i_0 i_1 \ldots i_{783}}\sum_{i_j=0, j\neq 352}^{8} a_0^{i_0} a_1^{i_1}\ldots a_{351}^{i_{351}} a_{353}^{i_{353}}\ldots a_{784}^{i_{784}}\right) x_{352}^{i_{352}}. \tag{10}$$

For $i_{352} = 0$, there are a very large number of terms collected into the constant – in fact, all terms that do not contain x_{352} are included in this term. Therefore, it is not entirely surprising that much of the prediction is collected into the constant when considering a particular slice of the function. What we show in the tables is the behavior of the function at a certain point in a particular direction.

We may repeat this for as many images or pixels as we like; in Table 4 we see the same polynomial generation algorithm applied to pixels 100 and 541.

We notice that in Table 4 as well as Table 3 that despite the activation function being applied three times, the polynomials we generate tend to be of low order. Additionally, with the polynomials we have seen, the constant terms are by far the largest of the polynomials. The network has in some sense already accumulated its prediction information into the constant term of the polynomial, since it is not making much use of the actual pixel information with such small coefficients on the x terms.

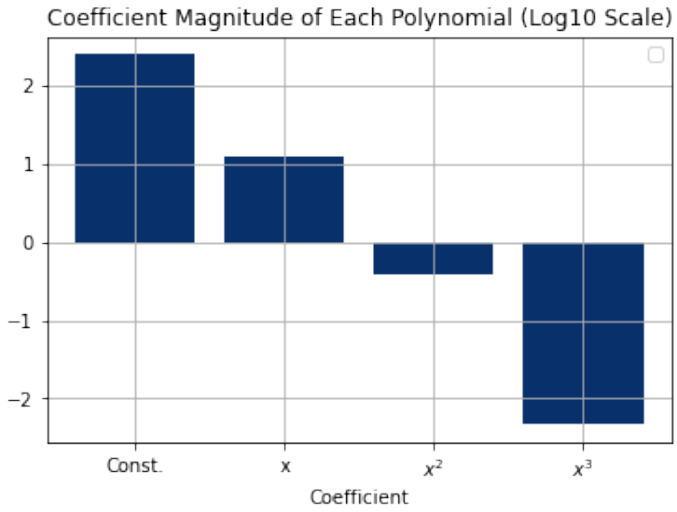

Figure 9: For several pixels in the MNIST image in Figure 8, we add up the sizes of the coefficients of the associated polynomials as listed in Table 3. We plot these sums on a log scale to demonstrate that the polynomials have a steep decline in order, by which we mean the magnitudes of the coefficients decease rapidly with order.

Table 4: Polynomials associated with each class prediction for the same image as Table 3, this time for pixels 100 and 541. For brevity, we only include the final 5 polynomials for each pixel. As before, polynomial coefficients less than .001 are not shown.

Pixel	Output Element	Polynomial
Pixel 100	Predicting 5:	$0.783 - 0.105x + 0.002x^2$
Pixel 100	Predicting 6:	$-7.651 + 0.163x - 0.002x^2$
Pixel 100	Predicting 7:	$1.576 - 0.063x + 0.002x^2$
Pixel 100	Predicting 8:	$1.798 - 0.077x$
Pixel 100	Predicting 9:	$17.113 - 0.202x + 0.001x^2$
Pixel 541	Predicting 5:	$0.787 - 0.098x$
Pixel 541	Predicting 6:	$-7.686 + 0.082x$
Pixel 541	Predicting 7:	$1.575 - 0.065x + 0.002x^2$
Pixel 541	Predicting 8:	$1.853 + 0.053x - 0.001x^2$
Pixel 541	Predicting 9:	$17.159 - 0.094x$

Table 5: Error rate for two models trained and tested on the MNIST data set of handwritten digits. Model 1 uses a standard LeakyReLU activation, while Model 2 uses the novel Piecewise Square activation. The model architecture is identical otherwise.

Model	Description	Error Rate (%)
Model 1	An MLP with three hidden linear layers and LeakyReLU activation function, trained with negative log likelihood loss.	$2.33 \pm 0.11\%$
Model 2	An MLP with three hidden linear layers and Piecewise Square activation function, trained with negative log likelihood loss.	$2.08 \pm 0.20\%$
Model 3	An MLP with two hidden linear layers and Piecewise Square activation function, trained with negative log likelihood loss.	$2.11 \pm 0.18\%$

We can actually make use of this fact to inform the architecture of the neural network. We know that since the coefficients on the non-constant terms are relatively small, the network no longer changes very much with respect to its input data by the time the data gets to the final layer. Therefore, prediction information has mostly already accumulated by the time the network gets to the last layer of the network. We can take advantage of our knowledge of what the network is doing by actually decreasing the depth of the network. We perform one final experiment here, where we decrease the depth of the network and see whether the network can still make good predictions on the MNIST data set. The results of this experiment are given in Table 5.

We see in Table 5 that the network trained with fewer layers actually slightly outperforms the larger networks, justifying our deductions about the behavior of the architecture given the size of the constant term in the polynomial. By this example, we have shown that *useful architectural insight can be gained by using the*

Piecewise Square activation function.

In this section, we have seen confirmation that not only is the piecewise square activation serviceable on real problems, but it is competitive with more standard activation functions such as LeakyReLU and can offer additional insight into architecture choices that other activation functions do not. We will explore the Piecewise Square activation function in greater detail when applying it to problems in electromagnetics later on.

3.4 Remarks on Activation Structure

We will make one final note here. The activation structure appears to be made more complicated than it needs to be to achieve the desired algebraic structure, considering the several additive terms in the pieces of the function in the left and right portions. With that in mind, we perform one brief experiment to show an alternative activation formulation we could use.

To that end, we define the following:

$$\text{Piecewise Square Activation, Simplified} = \begin{cases} 0.01x & x \leq 0 \\ x^2 & 0 \leq x \leq 1 \\ 2x - 1 & 1 \leq x. \end{cases} \tag{11}$$

This formulation removes some of the complexity in defining the function and putting it into code, though it gives up the C^1 property of the activation that will become important in later sections. We perform the same experiment on MNIST as with the other two models with this third activation function; the results are reported in Table 6.

While Model 3 reports a lower error rate than the other two models, we intend

Table 6: Error rate for two models trained and tested on the MNIST data set of handwritten digits. Model 1 uses a standard LeakyReLU activation, while Model 2 uses the novel Piecewise Square activation. The model architecture is identical otherwise.

Model	Description	Error Rate (%)
Model 1	An MLP with three hidden linear layers and LeakyReLU activation function, trained with negative log likelihood loss.	$2.33 \pm 0.11\%$
Model 2	An MLP with three hidden linear layers and Piecewise Square activation function, trained with negative log likelihood loss.	$2.08 \pm 0.20\%$
Model 3	An MLP with three hidden linear layers and Piecewise Square, Simplified activation function, trained with negative log likelihood loss.	$2.13 \pm 0.08\%$

to preserve the C^1 structure of our activation function as it will be important in later analysis in Section 6.

4 Electromagnetics

In the applied portion of the current work, also found in [54], our focus is on localizing an electromagnetic emitter sending a signal over water. While there has been significant work on using neural networks to perform range localization via acoustic signals in the underwater setting [15, 16, 33], as well as localization using electromagnetic signals in indoor environments [47], the authors are not aware of any work which attempts localization in an over-water environment, which is substantially more challenging. Herein we demonstrate how, in this setting, neural network performance *can be substantially improved by a mixing of knowledge of the physical properties of the system of interest with appropriate data selection and feature engineering*.

More specifically, we focus on the problem of localizing the source of an electromagnetic signal over water using only the resulting electromagnetic transmission, and possibly ancillary meteorological data. While it is fairly easy to calculate the angle at which a signal arrives at a receiver array (since one may simply rotate the receiver until the signal is strongest), it is much more difficult to determine the distance to the signal's origin. *Prima facie*, the problem is actually ill-posed, since one is unable to determine whether a signal comes from a powerful emitter that is far away, or a weak emitter nearby. However, the problem becomes possible when one makes use of the way the electric field scatters under certain meteorological conditions. Based on previous work on radio wave propagation over water [60], we believe it is possible to pinpoint the location of a radio transmitter under certain conditions within a few kilometers, using a specially trained regression neural network.

The most important of these meteorological conditions, and one of the key el-

ements of electromagnetic propagation in this over-water domain, is the existence of *evaporation ducts* [60]. As seawater evaporates, the water vapor forms a layer of humid air just above the sea level. This layer of humidity has a major impact on the propagation of electromagnetic waves. More precisely, it is the exact size of this evaporation duct which causes changes in the behavior of an electromagnetic signal. We can see an example of this in Fig. 10 - when the height of this evaporation layer is 20 meters, the electric field is much less oscillatory than when the duct height is 25 meters.

The other factor at play when analyzing signals and evaporation ducts in this setting is that in practice, the size of an evaporation duct is extremely difficult to measure [45, 62, 59]. This presents us with a coincidence of difficulties when trying to use electric field patterns to determine range - the factor which has one of the largest effects on the electric field is the exact factor which is the hardest to measure. One of the key findings of this work is that *while the existence of the evaporation duct (or other similar atmospheric phenomena) is essential, the precise height of an evaporation duct, while being crucial to the behavior of the electric field, is not needed to make range predictions based on the electric field if a deep neural network is appropriately trained*.

This problem is strange when it comes to deep learning in several ways. For one, unlike most problems in deep learning, we actually have access to the optimal model for the solution to our problem, known as a maximum likelihood estimator (MLE). This is already a remarkable departure from the usual setting in which one does deep learning. The other strangeness we see in the data set is extreme ambiguity in the data. What we mean by this is that there are input data points, corresponding to entirely different outputs, that are so similar that even the opti-

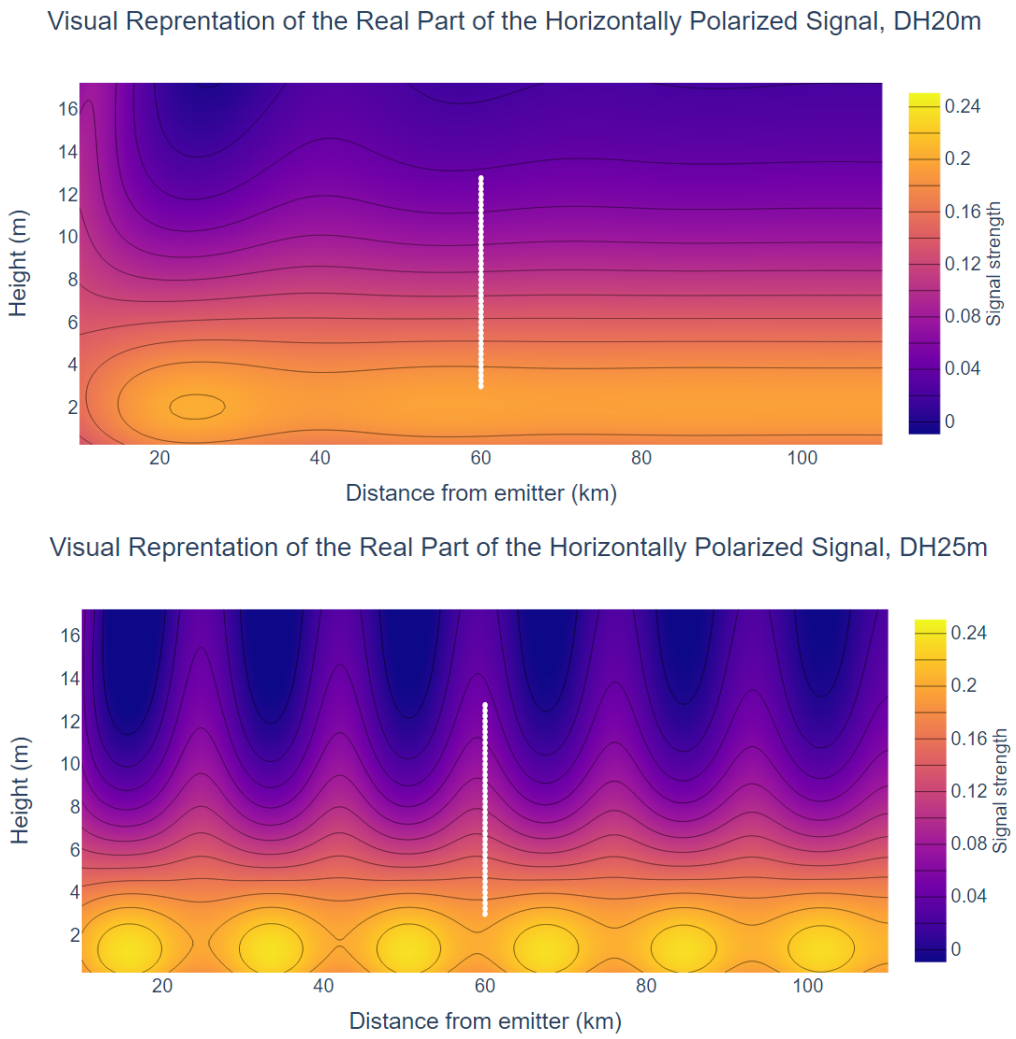

Figure 10: Electric fields resulting from evaporation duct heights of 20m (top) and 25m (bottom). Notice that the duct height has a substantial effect on the resulting electric field.

mal model to solve this problem cannot tell them apart. This leads to very unusual errors in the output for both a deep learning model and the MLE.

4.1 Deep Learning for Inverse Problems & Related Work

To place this problem in context, what we are attempting to achieve can be thought of as an inverse problem in partial differential equations, i.e., a problem in which one knows the output of the equation (in this case the electric field), and one wishes to find parameters of the equation which would lead to that output. There are many parameters which influence the electric field resulting from an over-water transmission – the height of the emitter, the power of the emitter, the size of an ambient evaporation duct, the presence of higher atmospheric ducts, the range at which one measures the electric field, and in practice, many more factors, which all have enormous impacts on the detected signal. Implicitly, these are all parameters in the partial differential equation that governs the behavior of the electric field.

Solving inverse problems in differential equations using neural networks is not new [4, 6, 40, 44]. However, rarely is this done with the little amount of data available to us in this setting. A typical example of an inverse problem in differential equations incorporating machine learning would be to discretize a space, give an algorithm examples of boundary conditions and solutions at all discretization points, and have the algorithm construct an equation which generates that solution. This task requires both large amounts of data and many points of measurement in the domain to be accurate. Unfortunately, such voluminous training data is not available to us for the current problem. In fact, the target application will likely only have a few independent collections of measurements, since each real-world experiment can take days to perform.

As we will see momentarily, our data set only has 80 sensors to capture an electric field spanning hundreds or thousands of square miles of ocean. It would be very difficult to invert the entire PDE with this in mind. Fortunately, we do not have to invert the entire PDE, we only have to recover one unknown parameter – the distance between the emitter and the sensor.

This problem of emitter localization is not new, in fact, work goes back to the 1940s [49]. This has remained an active area of research to the present. For a paper discussing emitter localization via triangulation with multiple receivers, see [13] or more recently, [46]. For an example of localization via a single sensor in an urban environment, see [37]. For emitter localization using a single sensor which moves, see [22]. The current work distinguishes itself by its explicit focus on the single-sensor, over-water setting, which as discussed, has much more complicated atmospheric conditions than other settings.

4.2 Generation and Preprocessing for Electric Field Data

The data we use for these experiments was generated using a MATLAB-based parabolic equation software tool, known as PETOOL[39]. Using this tool, we generate an electric field such as the one shown in Fig. 10. Fig. 10 shows only the horizontal polarization of the electric field, but the signal as generated by PETOOL has both horizontally and vertically polarized components. The simulated signal frequency is 5.8GHz.

4.2.1 Discretization and Sensor Simulation

In practice, the sensor that captures electromagnetic signals will have 40 receiver elements, spaced vertically at 25cm intervals from 3 meters to 13 meters, where

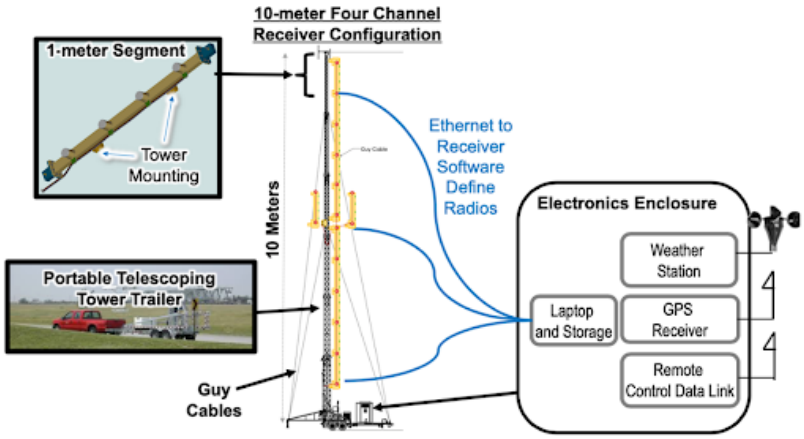

Figure 11: A diagram of the signal receiver. This diagram courtesy of Synoptic Engineering.

each sensor captures both horizontal and vertical polarization. A diagram of this receiver array is given in Figure 11. We replicate this signal measurement scheme in the simulated data by discretizing the electric field vertically at an interval of 25cm. The white lines in Fig. 10 represent the part of the electromagnetic field that is detected in a single measurement by both the real and simulated sensor. The horizontal spacing in the data discretization varies depending on the goal of the particular experiment. For initial experiments, the discretization is 20m, sampled from 10 to 110km.

For each measurement, the signal is received as 40 imaginary numbers for both the horizontal and vertical polarizations, so each data point is comprised of 160 real numbers. To construct a data point out of these measurements, we concatenate the horizontal and vertical components into a single 160-dimensional vector. A visualization of this process is given in Figure 12.

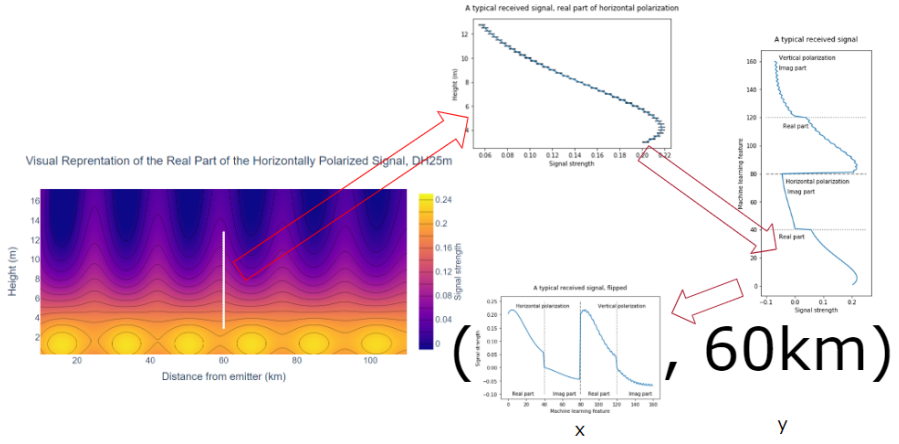

Figure 12: A visualization of our data organization process for each received signal. Our simulated sensors record 40 complex measurements of both the horizontally and vertically polarized portions of the signal. These 160 real numbers are concatenated into a single vector, which after additional transformations is used as the input vector for our deep learning methods on this problem. The range associated with that particular signal serves as the target output for that data point.

4.2.2 Normalization and Electric Field Rotation

Recall that the problem we are solving is to determine the range of an emitter based only on the resulting electric field. Now, importantly, if one is allowed to know the strength of the emitter, this problem is *far more trivial*. The reason is that the signal attenuates in a predictable way, therefore, one would only need to compute the norm of the received signal to determine the emitter range. In order to remove the emitter strength as a confounding variable for our network, we (separately) normalize the horizontal and vertical components of the signal to have norm 1 for every range.

There is another aspect that must be considered when it comes to electromagnetic signals, and that is the signal's phase. The signals in our data set are at the frequency of 5.8 GHz, naturally, with a 20m discretization of the electric field, the

phase of the signal changes at every discretization point. As we can see from Fig. 13, this leads to large incoherence in the data – the signal at 60km is very different from the signal at the very next discretization point at 60.02km. To solve this, we perform a simple, but perhaps nonobvious to anyone not trained in electromagnetic propagation, rotation of each imaginary data vector so that the first element is both positive and real. Put another way, given a signal vector

$$x = \begin{bmatrix} r_1 e^{i\theta_1} & r_2 e^{i\theta_2} & \dots & r_{40} e^{i\theta_{40}} \end{bmatrix}, \tag{12}$$

we subtract θ_1 from every angle, resulting in the vector

$$x_{rot} = \begin{bmatrix} r_1 & r_2 e^{i(\theta_2 - \theta_1)} & \dots & r_{40} e^{i(\theta_{40} - \theta_1)} \end{bmatrix}. \tag{13}$$

This transformation is done separately for the horizontal and vertical polarizations of the signal. A plot of several successive points after the rotation is given in Fig. 14. As we can see, successive data points are now reasonably coherent – a small difference in the range leads to a small difference in the electromagnetic signal. We will see later that this leads to substantial improvement in the performance of the network. We refer to data to which this transformation has been applied as *rotated* electric fields, and when this transformation is not applied, we refer to the electric field as *unrotated*. One of our principle investigations is to determine whether this rotational transformation is helpful when making range predictions. In theory, it should not make a difference, since the information provided to the network is unchanged. However, we will see later that a network trained on rotated data performs substantially better than one trained on unrotated data. The electric fields shown in Fig. 10 come from normalized and rotated data; without this, the signal

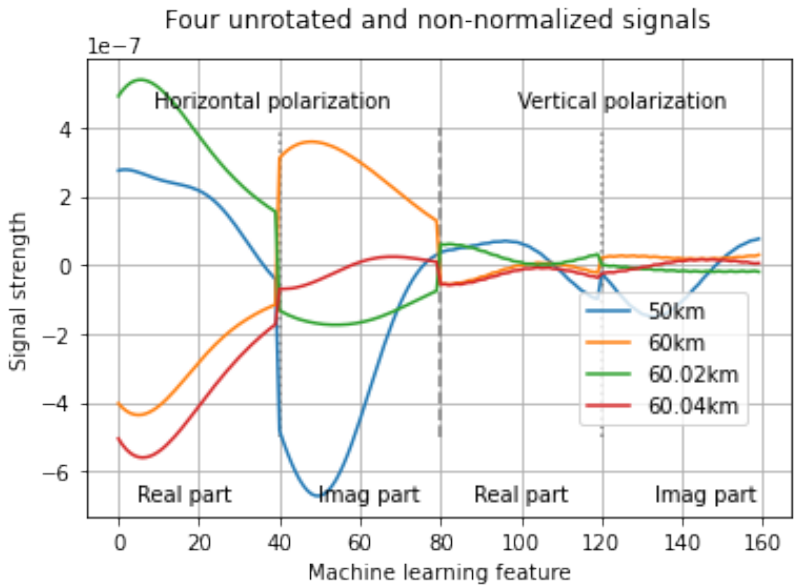

Figure 13: Four examples of received electromagnetic signals at specific ranges. Notice that three successive signals starting at 60km appear to be very different as a result of the phase.

would not appear to be a smooth surface.

4.3 Atmospheric and Experimental Parameters

4.3.1 Ducting Phenomena

In addition to the electromagnetic signal itself, we must also consider atmospheric conditions. The most important phenomenology in this problem space is the existence of what are known as *evaporation ducts*. As seawater evaporates, the resulting layer of humid air over the water can prevent electromagnetic waves from escaping upwards – this directs the signal horizontally, resulting in a significant improvement in the propagation of the electromagnetic wave [21]. Crucially, the larger this layer becomes, the greater the effect on wave propagation. For our purposes, we simulate 11 ducts, 20 and 25 meters high, with a spacing of .5m. Ducts of this size can occur in areas of the Caribbean and the Gulf of Mexico [19], so they

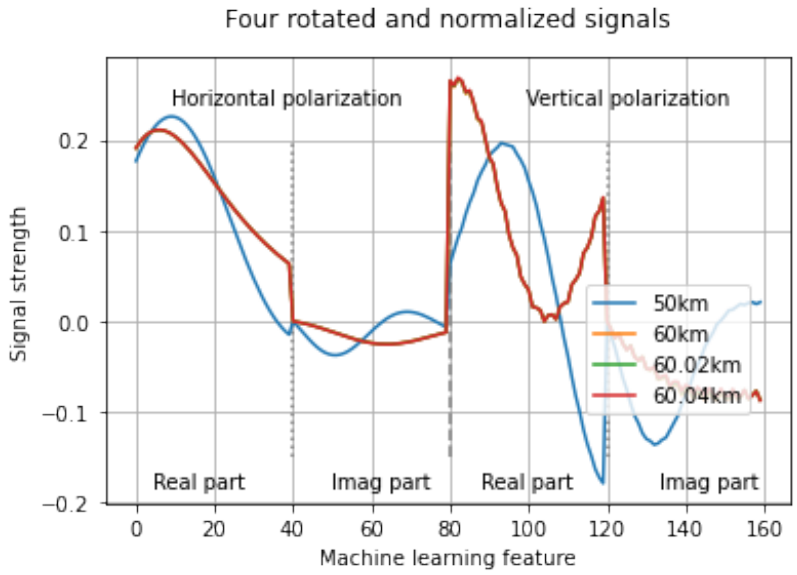

Figure 14: Four examples of the field patterns at the same specific ranges as in Fig. 13, rotated so that the first element of every field vector is positive real. Note that phase has been rotated out, leading to coherence in the signal not previously present.

are large enough to provide good wave propagation while still being physically realistic. Our training set is then 11 electric fields, each generated by a signal passing through a distinct evaporation duct.

The precise size and shape of the evaporation duct has an enormous impact on the electromagnetic field itself. We can see in Fig. 10 that there is a substantial difference in the two fields at duct heights of 20 and 25 meters. Additionally, evaporation ducts in practice are extremely difficult to estimate. The level of precision at which one can measure an evaporation duct is at least an order of magnitude larger than the level at which the size of the duct has a noticeable effect on the electromagnetic signal. This presents us with two major questions, both of which are essential to the current work:

1. Is information about the duct height, *measured at the level of precision that is possible in practice*, sufficient to help with range prediction?

2. Even *if* the neural network had access to perfect duct height information, to what degree (if at all) does this help with range prediction over having only the electric field?

Remarkably, the answers to these questions are ”no” and ”very little.” We will see later that regardless of the level of precision at which duct height information is provided to a neural network, it is always better to provide the network with *no* duct height information, allowing it to determine transmission range solely from the electromagnetic signal itself.

In addition to evaporative ducting phenomena directly over the ocean, another crucial part of the ducting phenomenology is higher layers of ducting known as *surface-based ducts*. These are additional ducting layers that form at higher altitudes of over 100m in some regions of the ocean. Qualitatively speaking, these layers bend the part of the signal broadcast upward back down toward the ocean surface, and therefore back down toward our real or simulated sensor. For practitioners in this space, this additional signal leads to greatly increased transmission range, as well as greatly increased detection range for incoming signals. For our purposes, surface-based ducts result in far more complex signals. A visualization of an electric field associated with a surface-based duct is given in Figure 15, along with an electric field resulting from an evaporation duct for comparison.

In the second image in Figure 15, the first 35km or so where the signal appears to be identical to the first image is known as the *skip zone*. This refers to the area where the signal has not yet been reflected back down to the ocean’s surface, and therefore this area is ”skipped” by the surface-based duct’s influence. After this, the received signal changes dramatically from the case of the evaporation duct.

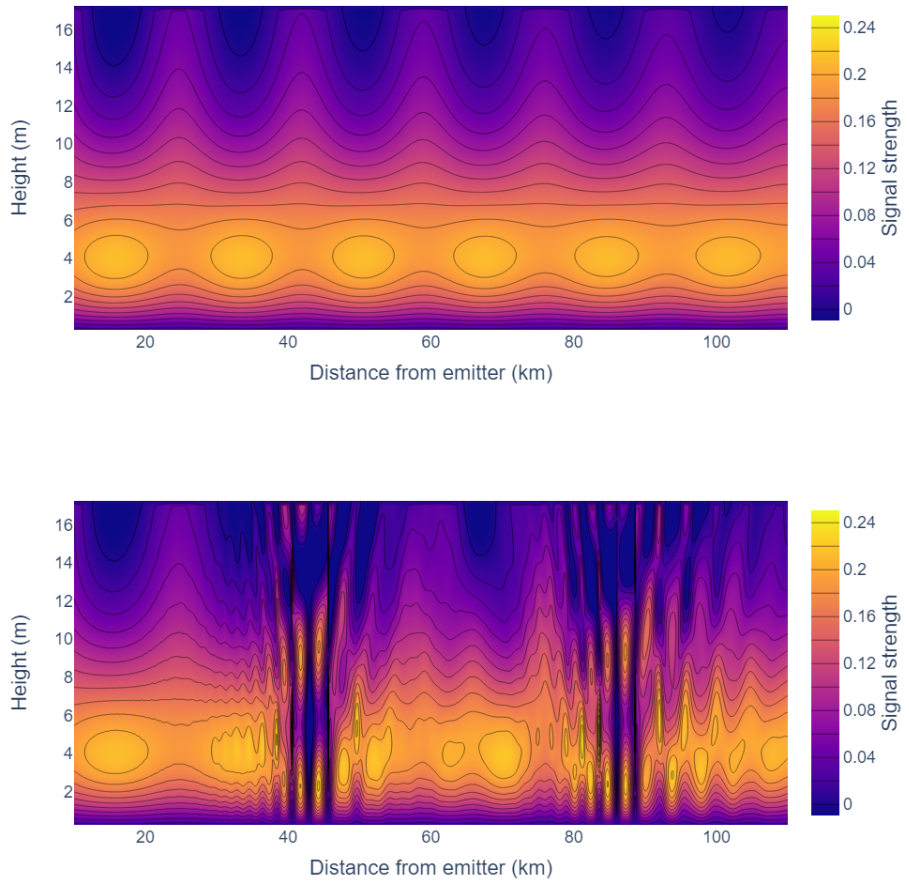

Figure 15: Top: an electric field associated with a 25m evaporation duct, like the one shown in Figure 10. Bottom: an electric field associated with a surface-based duct. Note the increased complexity of the electric field in this case.

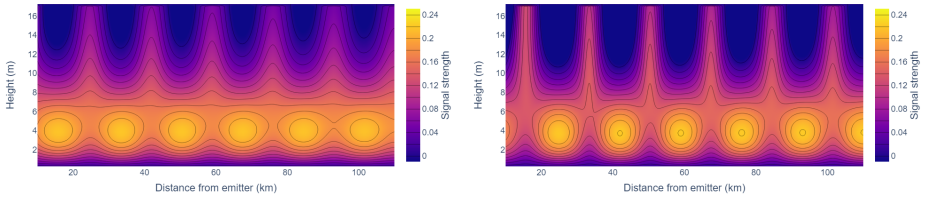

Figure 16: Left: an electric field associated with an evaporation duct height of 25m and a transmitter height of 1m. Right: an electric field associated with an evaporation duct height of 25m and a transmitter height of 10m.

4.3.2 Experimental Parameters

In addition to environmental conditions such as evaporation ducts, there are specific experimental parameters which have a noticeable impact on the received signal. Among these are the height of the receiver array and the height of the signal transmitter. A change in the receiver height is relatively easy to visualize in images such as Figure 10 – one may simply imagine the white dots representing sensors moving up or down within the image of the electric field. However, from the perspective of the sensor, this would certainly change the received signal in a very noticeable way. We will see examples of networks trained with both fixed and variable receiver height in our experiments. Typical values of the "receiver height" parameter are between 2.4 and 2.6 meters for these experiments.

The final experimental parameter we consider is that of transmitter height. This also changes the signal in a way which is noticeable to the sensor. In these experiments, we will typically consider a fixed transmitter height for any given experiment, typically either 2.5m or 5m. To show the extent to which this parameter changes the signal, an image of two signals with evaporation duct heights of 25m and transmitter heights of 1m and 10m are given in Figure 16.

4.4 Simulating Receiver Noise

There is one final consideration when it comes to the training data, and that is noise. PETOOL generates a noiseless electric field, to which we artificially add Gaussian noise representing the base level of noise in the sensor. In this signal processing domain, the signal-to-noise ratio is often referred to in decibels. Thus, instead of referring to the raw signal-to-noise ratio (SNR), we instead express it in decibels by writing

$$\text{SNR} = 10 \log_{10} \left(\frac{\mathcal{E}_S}{\mathcal{E}_n} \right), \tag{14}$$

where $\mathcal{E}_S$ is the total energy of the signal vector and $\mathcal{E}_n$ is the total energy of the noise. We refer to this as the SNRdb. $\mathcal{E}_n$ is a random variable, but following the law of large numbers, it will be concentrated around a value of $2N\sigma^2$, where N is the size of the vector under consideration. Rearranging, we see that one may choose σ^2 according to the formula

$$\sigma^2 = \frac{\mathcal{E}_s}{2N} 10^{-\text{SNR}/10}. \tag{15}$$

All that remains is to choose one's desired SNR and the range at which to sample a vector to measure its energy. As previously stated, we choose the vector at 60km, which is the middle of our training set, and a 25m duct height. The noise is added before normalization and rotation to mimic real-world data collection and processing.

In initial experiments, this Gaussian noise will have a standard deviation of $\sigma = 2.98 \times 10^{-9}$. This level of noise is chosen so that at a range of 60km and a duct height of 25m, the SNRdb is 40. 40db SNR is a mild level of noise, suitable for evaluating model feasibility. In later experiments, we will decrease the SNRdb

(thereby increasing the noise) to 20, which corresponds to a standard deviation of $\sigma = 2.98 \times 10^{-8}$, ten times the noise of the earlier experiments.

4.5 Training and Prediction

4.5.1 The Unrotated Electric Field

As a baseline for comparison, we train four neural networks on different data sets that are normalized but unrotated. The networks are trained on the following data sets:

1. Training data consists of 160 columns of the noisy, unrotated radio signal and one column of non-noised duct height.

2. Training data consists of 160 columns of the noisy, unrotated radio signal and one column of noisy duct height. The noise added to duct height was sampled from a Gaussian distribution with $.1$ standard deviation.

3. Training data consists of 160 columns of the noisy, unrotated radio signal and one column of noisy duct height. The noise added to duct height was sampled from a uniform distribution ranging from -2 to 2.

4. Training data consists of 160 columns of the noisy, unrotated radio signal and one column of duct height with all values set to 0.

Table 7: Parameters for neural network setup and training.

Network Architecture & Training Parameters	
Architecture	Fully connected MLP
Hidden Layers	5
Layer Sizes	$161 \to 100 \to 70 \to 60 \to 40 \to 20 \to 1$
Activation function	Leaky ReLU
Epochs	$10,000$
Dropout	None
Batch Normalization	None
Optimizer	Adam
Learning Rate	$1e{-3}$
Weight Decay	$1e{-5}$
Pytorch Version	1.8.1+cu102

4.6 Architecture and Training Scheme

These networks have identical architectures and are trained for $10,000$ epochs using an L2 loss function, where

$$Loss = \sum_i (y_i - \hat{y}_i)^2. \tag{16}$$

For all of our neural networks, we use a fully connected MLP architecture with five hidden layers. For these initial experiments, we use LeakyReLU as our activation as opposed to the Piecewise Square activation; we will see examples of electromagnetic experiments using Piecewise Square activation in Section 6. The layer sizes for the full network are $161 \to 100 \to 70 \to 60 \to 40 \to 20 \to 1$. We use Adam optimization from the PyTorch[42] optim library for our training. More details are given in Table 7.

Since we are using simulated data in these experiments, we are able to generate infinite training and testing data by taking the raw noiseless signal and adding new

instances of Gaussian noise whenever more data is required. To that end, we train our neural networks by adding new noise to the base electric field at every batch during training. This helps to prevent overfitting; since the network only ever sees any particular data point one time, it is never given an opportunity to learn any particular instance of noise. There are several robust techniques which can be used to prevent overfitting as well, however, with our method of data generation, overfitting was not enough of a concern in training to require their use.

4.7 Testing

"Training" and "testing," two of the most crucial aspects of constructing a good machine learning or statistical algorithm, are slightly more complicated in this setting than in most applications due to the "infinite" data we have access to in our data generation process. To reiterate our process, we generate one or more electric fields, apply Gaussian noise, then finally rotate and normalize the data to simulate real-world data collection and preprocessing. What makes training and testing complicated in this context is the existence of both atmospheric and experimental parameters as noted in Section 4.3, as well as the Gaussian noise we add to all data at every epoch of the network training process.

In many settings, a new sample set of data with new Gaussian noise could be considered a test data set, and indeed it is what we use for testing in some cases. And yet, a new set of data with new Gaussian noise is what is given to the network at every training epoch as detailed in Section 4.6. In that sense, a new set of Gaussian noise for the data is not truly "new," since the networks are trained to correctly fit arbitrary noise of a particular σ for any given electric field measurement during the training process.

Another way of testing data in this setting is to train a network with one set of atmospheric and experimental parameters, and then test on another set of conditions. For example, an interesting experiment may be to see whether a network trained on noisy electric fields with an evaporation duct height of 25m can accurately make predictions on noisy electric field data with an evaporation duct height of 24m. In some cases, this may more closely match a real-world training and testing scenario; the atmospheric conditions may change from day to day, and a network which is trained on data coming from an electric field under one day's weather conditions may be asked to make predictions on an electric field the next day.

For testing each network, we generate 100 instances of noise for each range and duct height pair in our discretization and perform a range prediction on each one using our trained network. Then for each range, we have 1,100 error values, for which we compute the RMSE.

The results of training these networks with no other preprocessing can be seen in Fig. 17.

4.8 Remarks

As we can see from Fig. 17, all three networks perform well when given the true duct height in testing. As one may expect, the network that was given the true duct height in training (blue) performs the best, followed by the network given the noisy version of the duct height (green), followed by the network which was not given the duct height at all (red). However, as the noise increases, the performance of the blue and green networks begins to degrade. In the second plot, the red network keeps exactly the same performance (since it does not make use of the

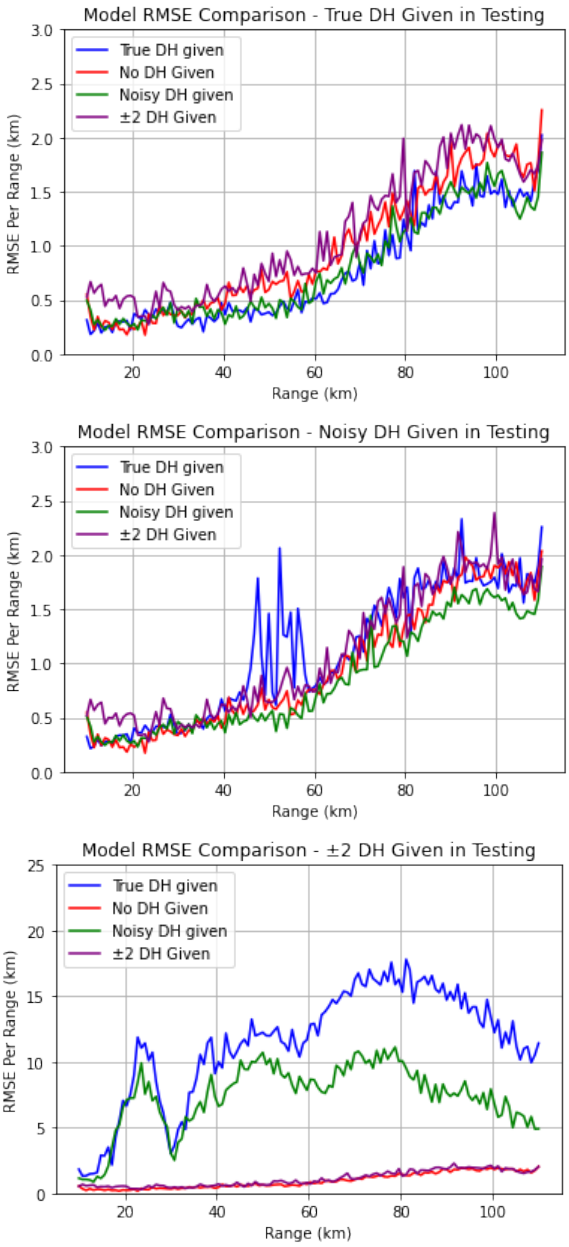

Figure 17: RMSE comparison of the four networks with an *unrotated* electric field. We see that the best networks in the (unrealistic) situation of having a perfect duct height measurement are the networks that were given accurate measurements in testing, but the network which generalizes the best to a realistic situation of very little duct height information is the one which was never given the duct height during training.

duct height), the green network maintains its superior performance over the red network, and the error in the blue network shifts above the other two, while exhibiting large spikes in its error at certain ranges. Most importantly, when all three networks are given a level of noise that is likely to be seen in application, only the network trained without duct height information has any hope of making accurate range predictions.

This leads us to our first major observation in these results: *For this problem, training the network on higher quality data than one can expect to see in practice will likely lead to disastrous prediction accuracy.*

As an application of this idea, we may consider the network trained on the duct height ± 2 (the purple network). One may be tempted to use this network on real data, since this is roughly the accuracy with which one can measure an evaporation duct in practice. If, in a real-world situation, the quality of measured duct height happens to be worse than what the network expects, the quality of the range predictions will go down accordingly. *Additionally, as we can see from Fig. 17 and Table 8, including such low-quality duct height information does not help the network in the first place!* The network finds much more accurate predictions when it relies solely on the electric field.

4.8.1 The Rotated Electric Field

We next train on the same electric field, but we perform the rotation discussed in Section 4.2. To simulate this in practice, we perform all normalization and rotation after the noise is added to the data.

1. Training data consists of 160 columns of the noisy, rotated radio signal and one column of non-noised duct height.

2. Training data consists of 160 columns of the noisy, rotated radio signal and one column of noisy duct height. The noise added to duct height was sampled from a Gaussian distribution with $.1$ standard deviation.

3. Training data consists of 160 columns of the noisy, rotated radio signal and one column of noisy duct height. The noise added to duct height was sampled from a uniform distribution ranging from -2 to 2.

4. Training data consists of 160 columns of the noisy, rotated radio signal and one column of duct height with all values set to 0.

As before, the networks are trained for $10,000$ epochs using an L2 loss function, with the same network architecture and Adam optimization as shown in the Table 7. Additionally, noise is added at every batch to prevent the network from learning any particular instance of noise. To evaluate each network, 100 new instances of noise for each of the 11 duct heights we train with are given to each network. We then compute the RMSE of the 1,100 trials at each range and report this as the test error for that range. The results of this testing are given in Fig. 18.

We see similar phenomenology in this case as in the unrotated case. When all four networks are tested on data containing the true duct heights, by far the best network is the one that was trained on the true duct heights as well. However, this network does not generalize at all to a situation where it is not given accurate duct height information. Since evaporation ducts are impossible to measure with the perfect accuracy required to give the most accurate predictions, a network such as this is unable to be used in practice. Thus, the network that has the best generalization is once again the network trained without any knowledge of the duct height.

Crucially, however, we see in Table 8 that incorporating signal rotation into the preprocessing step decreases the RMSE by 27.1% for the network trained without duct height. The error reduction is remarkable, and perfectly exemplifies the power of proper data transformation prior to training. This transformation is remarkably simple, and yet it leads to a substantial decrease in test error. This demonstrates the importance of incorporating knowledge of the physical system into feature engineering. The physical property of the phase of the signal led to a confounding of the network, so by removing the phase as a factor, not only is the data cleaner, but prediction performance improves.

Table 8: RMSE (km) for every network when evaluated on 100 electric fields for each of the 11 duct heights in the training set with different noise realizations. As the amount of duct height noise in the test data increases, the network trained on no duct height information retains its performance, while the performance of the other networks degrades rapidly.

Unrotated

	True DH	Noisy DH	± 2 DH	No DH
True DH – Testing	.914	.931	1.225	1.130
Noisy DH – Testing	1.238	.991	1.227	1.129
± 2 DH – Testing	12.246	7.789	1.233	1.128

Rotated

	True DH	Noisy DH	± 2 DH	No DH
True DH – Testing	.722	1.197	1.005	.824
Noisy DH – Testing	1.532	1.242	1.007	.822
± 2 DH – Testing	17.046	10.496	1.006	.822

4.9 Discussion

Surprisingly, the network trained without duct height included at all had a roughly equivalent test error to the other two networks. This confirms the hypothesis that all the duct height information is included in the signal itself. Additionally, the

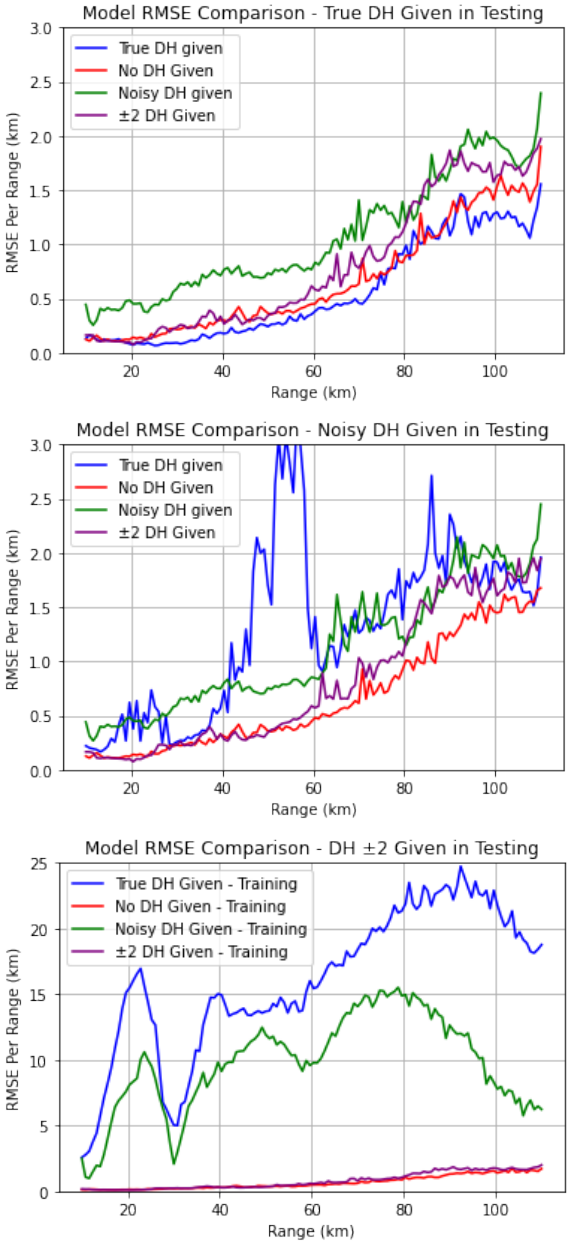

Figure 18: RMSE comparison of the four networks with a *rotated* electric field. Each network is tested on 100 new noise realizations for every range and duct height, and the RMSE of the 1100 trials at each range is displayed. Note that the network trained without any information on duct height has the best generalization to the situation where accurate duct height measurements are impossible.

network trained without any duct height information actually does better for most ranges than the network that was given the correct duct height. The most likely explanation is that when the network is given a duct height, it will spend network parameters trying to make use of it, when those parameters could be spent more productively interpreting the signal itself.

This becomes clear when one considers the other two plots in Fig. 18. When tested on less duct height information than what it was trained on, the performance of each network drastically decreases. This is most obvious when considering the loss plot when the networks are given low quality duct height information – the ones trained to make use of the duct height completely fail. This proves that they are using the duct height to make predictions when in reality, it is unnecessary.

It is also important to note here that in Fig. 18, the blue and green graphs have access to the same electric field information than the red graph – in theory, they could set the duct height variable to 0 in the first layer and find the exact same network. However, given the enormous errors when duct height information is removed, they clearly do not do this.

4.10 Further Evaluation

4.10.1 Benchmark Model Comparison

In order to find a more absolute way of determining the quality of the predictions, we compare our best network, trained on no duct height information and rotated data, to a maximum likelihood estimator (MLE) such as the one given in [43]. Crucially, this model knows all the physics of electromagnetic propagation over water; furthermore, this estimator is known to be optimal in a simulated situation such as

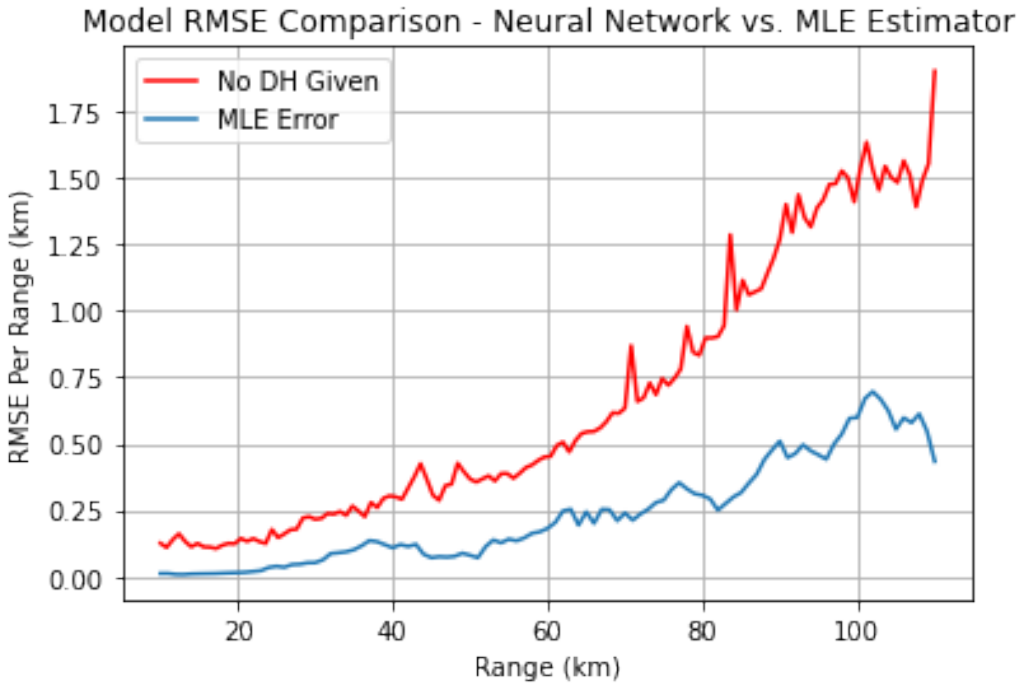

Figure 19: A model comparison between the neural network and the optimal MLE estimator. It is important to note that the blue line has precise and full knowledge of the physical system being measured. The fact that a relatively generic fully connected network comes close to the optimal solution with full physics knowledge is quite promising.

this, where the assumed physics of the model perfectly matches the physics that generated the data. Thus, a neural network cannot outperform the MLE in the simulation case, rather, the MLE error serves as a lower bound which the neural network attempts to approach.

We see in Fig. 19 that the difference between the neural network and the theoretically optimal MLE estimator is entirely reasonable. Especially for longer ranges, the neural network is within a factor of three of the optimal prediction. Additionally, there are no ranges for which the error is more than a few kilometers. The difference is remarkably low when one considers that the MLE knows the physics of this system perfectly, while the neural network, of course, knows none.

One might wonder, if the error of the MLE is lower than the neural network, why one would attempt to use a neural network model at all. There are two reasons for this. One, the MLE works by searching over all unknown parameters of the system and determining, based on the electric field, which combination of parameters is the most likely. One then extracts the range as one of these parameters for localization estimate. When the number of unknown parameters in the system is high, as it would be in practice, the computation cost can be too prohibitively high to use in a real-world scenario. Additionally, the estimator must be re-run every time one wishes to do a localization estimate. The advantage of a neural network here is that nearly all the computation cost comes at the front end when the network is trained, while model evaluation in practice is extremely computationally cheap. The other reason to prefer a neural network model for this problem is that the MLE is extremely sensitive to model mismatch, that is, when the physics of the real system do not match the assumed physics of the model. When the signal passes through a more complicated atmospheric system, for example, if there is a higher ducting layer above the standard evaporation duct, the MLE must either perform very badly or use enormous computation time to take into account the more complicated physics. Conversely, the neural network simply needs more data to account for more complicated physical situations.

5 Short-Range Experiments

5.1 Data Ambiguity

One of the most challenging aspects of this problem is the notion of data ambiguity. Let us consider for a moment the overall scheme of this electromagnetic problem. A transmitter broadcasts an electromagnetic signal, the receiver detects and measures the signal, and an algorithm attempts to determine the distance to the receiver. The algorithm is able to even attempt this because as the signal passes through the atmosphere, it is scattered in a particular way which allows the algorithm to gain information from the shape of the received signal. Crucially, if this entire process were happening in a vacuum, this would be *impossible*. A (normalized) signal would look identical at every range, leading to no information contained in the shape of the signal that an algorithm could use to determine the distance to an emitter. While this is not the case we study, it is useful to keep this in mind because even in the case in which we work, in which a signal passes through over-water evaporation ducts, some of this ambiguity remains.

This ambiguity is best demonstrated in the behavior of the Maximum Likelihood Estimator [43]. We see in Figure 20 a plot of likelihoods of each potential range when the MLE is given an electromagnetic signal and asked to determine the distance to the transmitter. The MLE works by considering a grid of ranges, in this case a grid in which the grid points range from 10km to 110km and have 20m spacing. For each grid point representing a distance to the transmitter, the MLE determines the probability that the signal came from that distance. We see in Figure 20 that this probability plot does not have a single probability peak representing the true distance. Rather, multiple peaks develop. The meaning of this plot is that

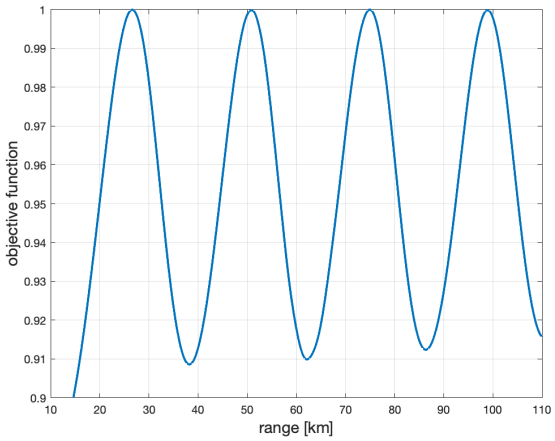

Figure 20: Given an electric field measurement coming from a distance of 50km, for every range in the spatial discretization, we ask the Maximum Likelihood Estimator to determine the likelihood that this signal came from that range and plot the normalized probabilities. We see that while there exists a probability peak at 50km representing the true answer, there are other peaks at 24, 76, and 100km which are also very likely answers. Image courtesy of Rick Brown, WPI.

the phenomenology of this problem is such that the electric field somehow nearly repeats itself, that is, data points 20km away from each other look so similar that even the optimal range estimator in the MLE can easily confuse them.

What makes this a real challenge is the fact that noise is added to the data prior to its evaluation by any of our models. We see in Figure 20 that the while the peak at 50km is the highest, and therefore the most likely, the other candidate ranges associated with the peaks at 24, 76, and 100km have very close to the same probability of being the correct ranges. When noise is added to the signal, the prediction may ”jump” to the next peak, leading to bi- or tri-modal predictions as shown in Figure 21.

This tell us something fascinating about the manifold on which our electromagnetic signal data lives in 160-dimensional space – in some sense, the manifold has ”folds,” where points which are far away in the parameter space, e.g. points which have very different ranges associated with them, are very nearby in the data space.

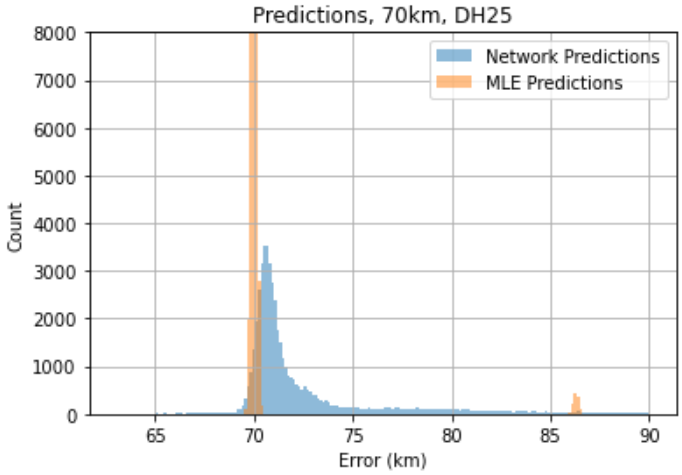

Figure 21: Comparison histogram of the MLE and the neural network model when both are tested on electric fields which are 70km away from the transmitter. As we can see, even the optimal MLE makes ambiguity errors on data at this range and this level of noise. The neural network makes ambiguity errors as well, but in a qualitatively different way. The "spreading out" of the predictions is due to the fact that a neural network is a continuous function, while the MLE is not.

For this problem, data points which may be 50km apart in the parameter space may be close enough in the measurement space that even with a reasonable level of noise, the MLE and neural network both confuse them.

5.1.1 Prediction Ambiguity for Neural Networks

We clearly see the same phenomenology develop in the case of the neural network, and we perform a new experiment to demonstrate this. We train a new neural network, again on an evaporation duct height of 22m and on the ranges 10 to 110km. We train the network for 10,000 epochs using Adam optimization as before. Crucially, we train the network to make predictions on data with a SNRdb of 40, which is the lower level of noise. We want to see what happens when we test the network on a *larger* level of Gaussian noise than it was trained on.

We examine the behavior of the network when it is trained on an electric field

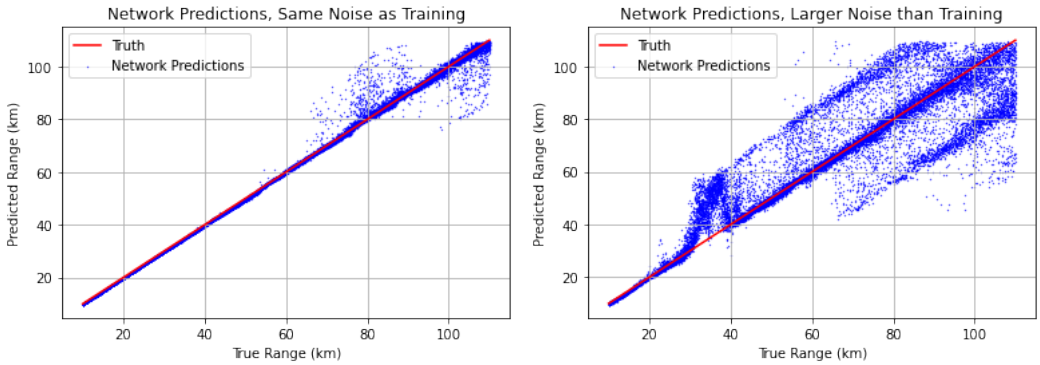

Figure 22: Left: the predictions made by a neural network trained with artificial Gaussian noise corresponding to 40db SNR, tested on data of the same level of noise. Right: The predictions made by this same neural network, trained on noise of 40db SNR, when the testing data has its noise level quadrupled. We can see that prediction ambiguity, which was already present at the lower level of noise, causes the network to make extreme errors when the noise is increased.

with Gaussian noise with an SNRdb of 40, tested on the same electric field with the noise level quadrupled. One would expect the error in the predictions to increase, and indeed that is what happens. What is unexpected is the precise *way* in which the error increases. We see the result of this experiment in Figure 22.

What is shown in Figure 22 is a set of range predictions made by the neural network on an electric field with new Gaussian noise, plotted against what the true ranges are for those electric field measurements. If the network were making perfect predictions, the true range and the predicted range would be exactly equal, and the plot would be a straight line of blue dots which follows the red line.

The first plot shows the set of predictions when the size of the Gaussian noise in the test data is exactly equal to the size of the Gaussian noise learned by the network at training time. We see that even in this first case, for long-range predictions, the network begins to make ambiguity errors – the network occasionally predicts up to 100km when the true range is 80km, and vice versa.

The second plot in Figure 22 shows the predictions made by the network when

the level of Gaussian noise in the testing data is four times the level of Gaussian noise in the training data. Here, the number of ambiguity errors increases tremendously. Additionally, the network begins making these errors at shorter ranges, though they are especially common at ranges greater than 60km. The network confuses electric field measurements associated with a range of 60km with electric field measurements associated with 80km. At very long ranges of over 100km, the network has a very hard time determining the true range. It occasionally predicts around 60km, which is "two peaks away" from the correct range prediction, to borrow language from the behavior of the MLE.

5.1.2 Ambiguity Analysis

We explain this difficulty with the illustration in Figure 23. To use the example in the previous experiment, imagine that the point $(0, 0)$ corresponds to the electric field measurement at a range of 60km, and the point $(2, 0)$ corresponds to an electric field measurement at a range of 80km. The blue and orange clouds of data points correspond to a noisy version of those measurements. In the left plot in Figure 23 corresponding to smaller noise, the two groups of measurements are distinguishable relatively easily – the network would have little issue mapping blue points to 60km and orange points to 80km correctly. In the right plot, with the noise increased, it is no longer possible to achieve the task of making predictions on the two classes consistently. It is particularly interesting that in order to think correctly about this range ambiguity in our regression problem, it is necessary for us to invoke the language of classification methods.

There are multiple ways to address this ambiguity concern, though none are simple. The first notable way is inspired by the shape of the prediction histogram

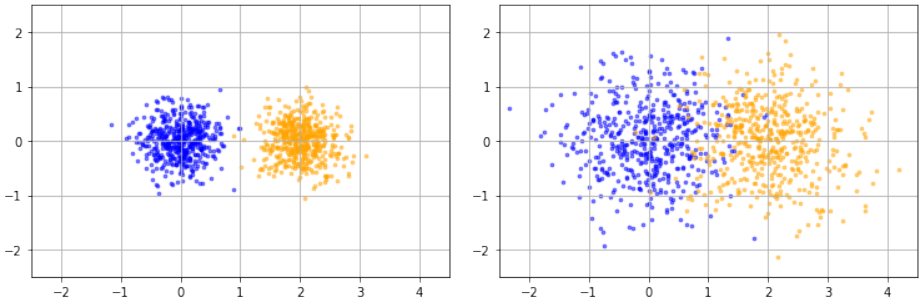

Figure 23: Left: a scatter plot of two sets of 100 samples of 2d Gaussian noise with standard deviation $\frac{1}{3}$ with mean $(0, 0)$ and $(2, 0)$. Right: the same Gaussian noise, with the standard deviation multiplied by 3. We see the mixing that happens when the level of noise is increased – this is analogous to mechanism by which ambiguity errors are made by both the neural network and the Maximum Likelihood Estimator.

in Figure 21. We notice that while there is a large "spread" of predictions, in the sense that the network predicts many different ranges from 70 to 90km, by far the most *frequently* predicted value is clearly very near to the correct range of 70km. One could imagine in a real-world scenario taking many measurements, generating a histogram of range predictions like the one in Figure 21, and using the most common prediction from the neural network as the true range prediction. This solution would require a series of predictions to happen relatively quickly, which may or may not be possible depending on the use case in practice. One could consider this the "vertical slice in time" solution.

Another possible way to address the range ambiguity is to consider a series of measurements over a period of time. Imagine in a real-world situation that a signal is received, and the neural net predicts the range of the signal to be 65km. A minute later, another measurement is made, and the neural net predicts the signal to be coming from a range of 65.5km. This continues with the network making range predictions between 65 and 70km. Finally, another measurement is made a short time later, and the network predicts the range to be 90km. One could imag-

ine ignoring this range prediction based on the physical principle that a transmitter cannot teleport 20km in a short span of time. One could consider this the ”horizontal slice in time” solution, since it relies on making predictions over a span of time.

5.2 Experimental Results

We have shown in the previous section that our network's generalizability is improved when it is not made to rely on being given ambient weather data, but rather relying strictly on the electric field data. However, an improvement from one set of data to another is not a measure of absolute quality. Put another way, we want to re-examine the model not from a perspective of how atmospheric data changes the performance, but from the perspective of asking whether the predictions given by the model are ”good” in an absolute sense.

In many settings of machine learning, this can be a difficult question to answer. In this setting, a key tool that we have, which is not available in classical deep learning problems, is the optimal method for range localization in the form of a maximum likelihood estimator (MLE). We are able to actually compare the performance of the MLE against our neural network method to see how well the network is performing compared to what is theoretically optimal.

One may ask why, if an optimal method exists, one would want to use deep learning in the first place. The reasons for this are twofold: one, the MLE requires a very accurate model of the physics of the problem. While this is possible for data generated via simulation, anyone who has ever worked with atmospheric data knows that such a model is very difficult, if not impossible, to obtain in practice. Second, the MLE works by having access to all possible electric field measurements

corresponding to all possible configurations of the atmosphere through which the signal passes. It then compares a given signal to all possible signals, determines which is most likely, and then makes a prediction based on this comparison. Doing this in practice would require a prohibitively large amount of data and computation time, thus, a deep learning method may be the correct practical choice due to its quicker evaluation time.

We have a number of experiments that demonstrate the accuracy we are able to achieve on simulated electric field data. We start with the simplest case and work our way up to larger experiments.

5.2.1 Experiment 1 – Single Duct Height

What we show here is a histogram of the errors of the MLE and neural network methods when both are evaluated on 50,000 data points. These data points are generated by taking a base measurement of an electric field, say, the field with an evaporation duct height of 21m measured 27km away from the transmitter. We add 50,000 realizations of Gaussian noise to this measurement, then evaluate each method on these 50,000 measurements.

Our next experiment is to train a neural network on a single electric field generated by PETOOL associated with a single evaporation duct height; in this case the duct height we train on is 22m. As we have demonstrated, performing the data rotation as detailed in Section 4.2.2 significantly decreases the overall prediction error, thus, we will be performing this transformation on all experiments moving forward.

As before, our network architecture is that of a fully connected MLP with identical layer sizes as the earlier experiments. We train the network for 13,000 epochs

Table 9: Parameters for neural network setup and training – Experiment 1.

Network Architecture & Training Parameters	
Architecture	Fully connected MLP
Hidden Layers	5
Layer Sizes	$161 \to 100 \to 70 \to 60 \to 40 \to 20 \to 1$
Activation function	Leaky ReLU Activation
Epochs	$13,000$
Dropout	None
Batch Normalization	None
Optimizer	Adam
Learning Rate	$1e{-3}$
Weight Decay	$1e{-5}$
Pytorch Version	1.8.1+cu102

using Adam optimization as before.

5.2.2 Experiment 2 – Train on One Duct Height, Test on Many

In the setting of making predictions on data associated with atmospheric conditions, it is generally not expected that any two experiments will have the exact same set of atmospheric parameters. To use a familiar example, the evaporation duct height will change from day to day. A realistic workflow in this setting would be to gather electromagnetic data on one day, train a neural network model to make range predictions based on this data, then use that trained network in the field to make predictions on a later day. If the atmospheric conditions at time of testing or actual field usage are different than those from which the training data was gathered, the network will make prediction errors. In this section, we begin answering some of these questions.

We devise an experiment to get a sense of the performance of the neural network in this setting. We are able to use the same trained network as in Experiment

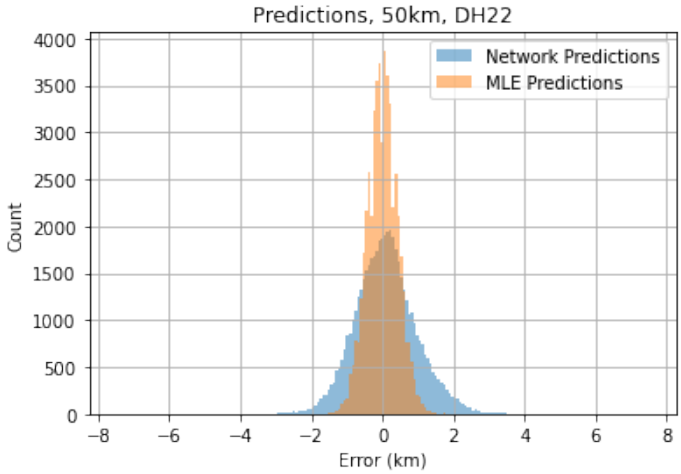

Figure 24: A histogram of $50,000$ prediction errors when a network is trained and tested on a single evaporation duct of height 22m.

1. This network has been trained on signal data associated with an evaporation duct height of 22m. For this experiment, we test this network on various duct heights at various distances from the training duct height. The testing duct heights will vary between 21.4 and 22.6m, spaced every 10cm and excluding the original training duct height of 22m. We refer to the disparity between training and testing duct height as model mismatch.

We would also like to compare the neural network output in this experiment to the Maximum Likelihood Estimator as before. We must be careful about making using "training/testing" language in this comparison, since the MLE is not "trained" as such. Rather, the MLE works by making use of a large table of potential electric fields at various ranges and associated with various duct heights. Given a candidate electric field measurement on which to make a range prediction, the MLE searches over its table of electric fields, selects the measurement most similar to the candidate, then reports the range associated with that measurement. What "model mismatch" means for the MLE is for the user to only allow the MLE to search over electric fields associated with a particular duct height – in this case,

we only allow the MLE to consider electric fields associated with the duct height of 22m.

The results of this comparison are given in Table 10, Figure 25, and Figure 26.

Table 10: Comparison of model performance between the Maximum Likelihood Estimator and the neural network models when both models experience mismatch between the training data set (22m) and the testing data set. As the mismatch grows, both models experience degradation of the model performance. Interestingly, the neural network experiences this degradation in a gentler way – it is less sensitive to model mismatch, though the differences in mean predictions are generally small.

Test DH	Network Mean	MLE Mean	Network Std Dev	MLE Std Dev
21.4m	48.78	47.78	1.85	0.57
21.5m	48.84	48.10	1.56	0.53
21.6m	48.99	48.44	1.35	0.49
21.7m	49.19	48.80	1.18	0.46
21.8m	49.46	49.18	1.10	0.45
21.8m	49.76	49.58	0.98	0.43
22.1m	50.46	50.45	0.88	0.40
22.2m	50.85	50.93	0.87	0.40
22.3m	51.28	51.44	0.81	0.39
22.4m	51.72	51.99	0.80	0.39
22.5m	52.20	52.57	0.79	0.39
22.6m	52.71	53.19	0.75	0.39

5.2.3 Experiment 3 – Train on Many, Test on Many

Experiment 3 is the largest we have seen thus far. In this experiment, we will train the neural network to fit data coming from 15 duct heights and 10 receiver heights. We are asking the neural network here to fit data coming from 150 electric fields. Instead of cleanly discretizing the data in the duct height variable as before, we generate 15 random duct heights to include in our data set. As before, these randomly sampled duct heights lie between 20 and 25m, while the randomly sampled

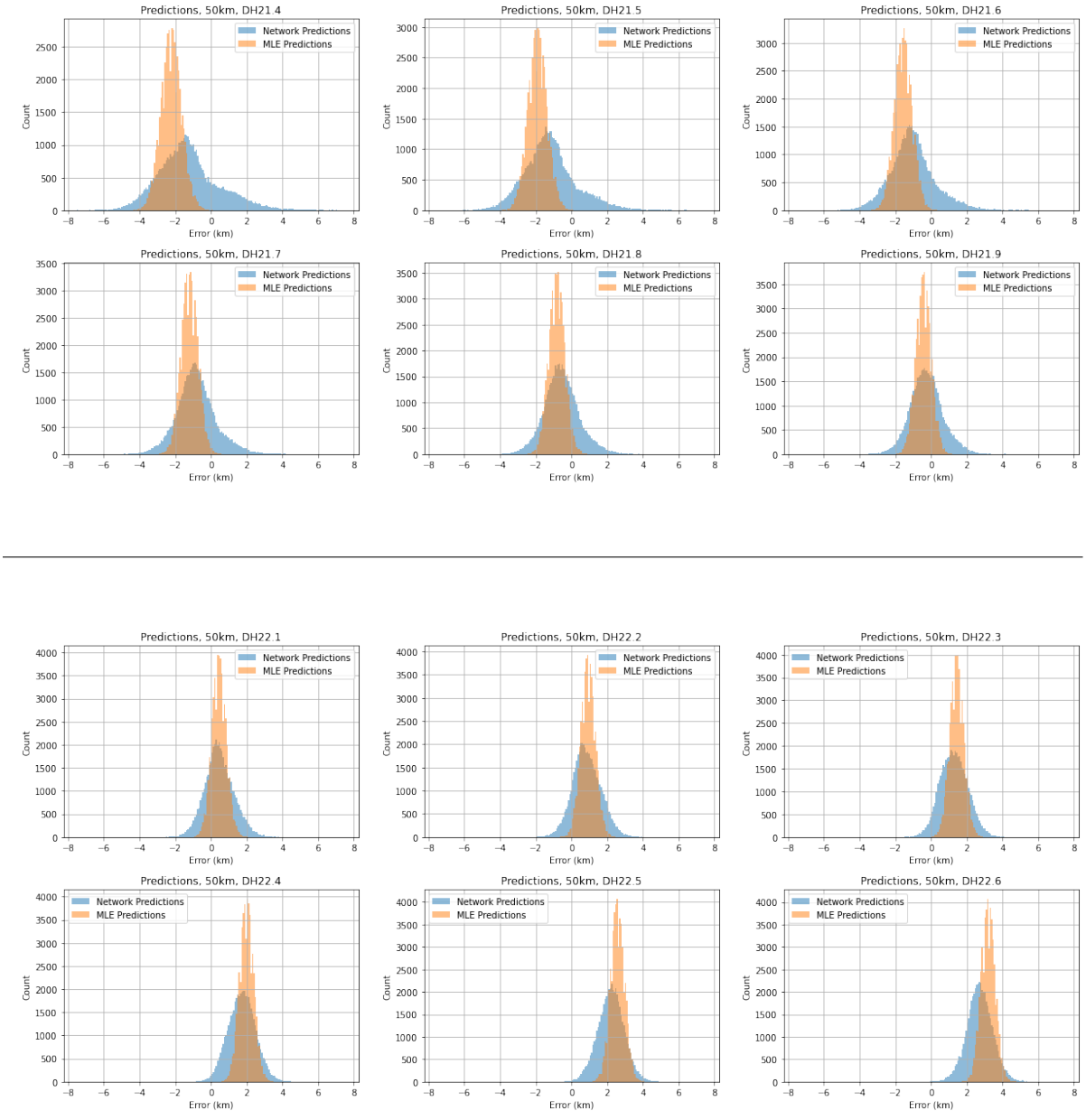

Figure 25: Prediction Error when a neural network is trained on electric field data associated with an evaporation duct height of 22m, then tested on various duct heights between 21.4 and 22.6m. Top: Prediction error for test data lower than 22m. Bottom: Prediction error for test data above 22m.

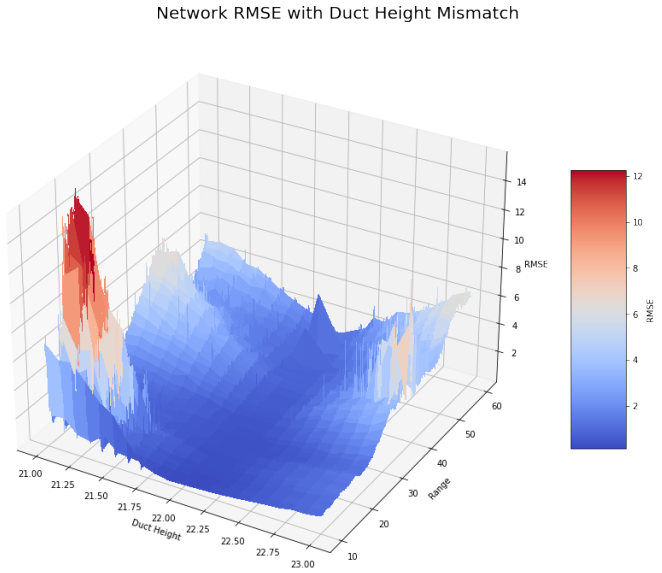

Figure 26: RMSE when the network trained on data associated with an evaporation duct height of 22m is tested on data associated with duct heights 21 through 23. As we can see, there is a gradual decrease in the quality of predictions as the network moves away from its learned duct height.

receiver heights lie between 2.4 and 2.6m.

For this experiment, the ranges we consider are between 5km and 60km, and our range discretization spacing is 20m. Therefore, each of the 150 electric fields has 2751 measurements associated with it, each of 160 dimensions. Our architecture is identical to previous experiments, and we train this network for 12,000 epochs using Adam optimization.

To develop a robust test of this network, we discretize the parameter space of receiver height and duct height and generate a new electric field for each pair. The duct heights we choose for testing lie on a grid of 31 points between 20 and 25m, while the receiver heights are still generated randomly between 2.4 and 2.6m. Thus, there are 155 electric fields considered in this test, each of 2751 range measurements. On top of that, for each of the 155 noiseless electric fields and each range, we generate 15 samples of Gaussian noise with an SNRdb of 20. The result

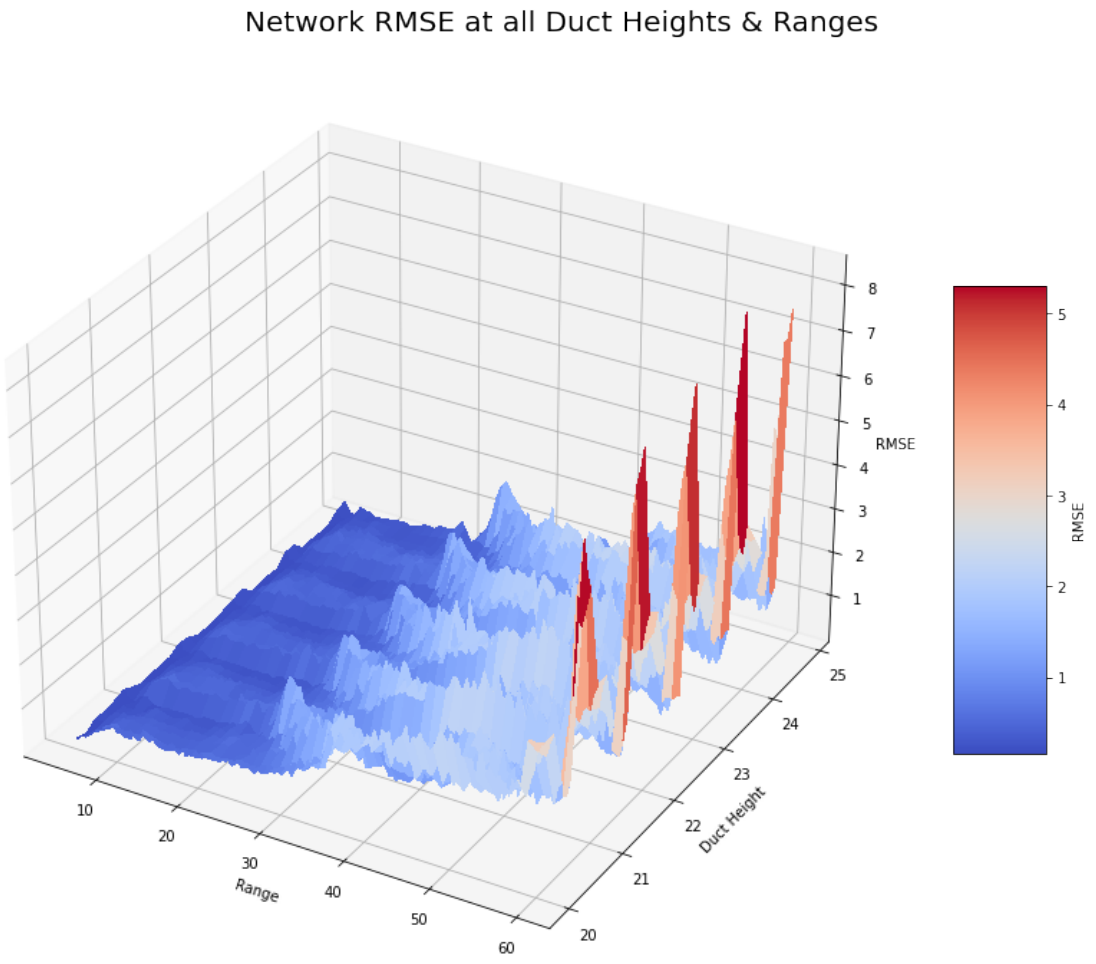

Figure 27: A plot of the RMSE of the neural network predictions for all ranges from 5 to 60km and duct heights from 20 to 25m. We note the regime changes at 30 and 60km where the error increases.

of this is that for each range and duct height pair, there are 75 samples of possible electric field measurements corresponding to different receiver heights and different noise realizations.

We are interested in the error of the network for range and duct height pairs, so for each of these pairs, we compute the RMSE of the 75 samples. The plot of this RMSE surface is given in Figure 27.

6 Real-World Application of Piecewise Polynomial Activation

We have seen that networks with a standard LeakyReLU activation can perform comparably to the MLE in many situations. To be precise, the deep learning methods with LeakyReLU activation have successfully found a set of functions which map electric fields to transmission ranges in a way which approaches optimality when compared to the maximum likelihood estimator. As has been our focus in the theoretical portion of this work, we now ask the question: what function has the network found? What are its properties? With the tools provided to us by using a piecewise polynomial activation, we are ready to answer some of these questions.

6.1 Training and Testing on Electromagnetic Data

We begin by replicating some of our smaller experiments with Leaky ReLU, and comparing the results with the Piecewise Square architecture to the more standard LeakyReLU architecture. As summary of the architecture is given in Table 11. The architecture is identical except for the activation function to provide a direct comparison of the performance of Piecewise Square activation with LeakyReLU.

For this experiment, we replicate the conditions of the first experiment listed in Section 5.2.1. That is, we train two networks on an electric field of duct height 22m, with a single receiver height for simplicity. We train both networks for 25,000 epochs using Adam optimization.

For testing, we use the same original noiseless electric field associated with the evaporation duct height of 22 meters, then for each candidate range, we gener-

Table 11: Parameters for neural network setup and training.

Network Architecture & Training Parameters	
Architecture	Fully connected MLP
Hidden Layers	5
Layer Sizes	$161 \to 100 \to 70 \to 60 \to 40 \to 20 \to 1$
Activation function	Piecewise Square Activation
Epochs	$25,000$
Dropout	None
Batch Normalization	None
Optimizer	Adam
Learning Rate	$1e{-3}$
Weight Decay	$1e{-5}$
Pytorch Version	1.8.1+cu102

ate 200 instances of Gaussian noise to create 200 candidate measurements for each range. We have each network make predictions on each of the 200 noisy measurements, then compute the RMSE of these 200 measurements and display the RMSE for each range as our test error for that range.

The testing results of this experiment are given in Figures 28 and 29. Our results here are consistent with the MNIST results in Section 3.2 – the Piecewise Square activation achieves results comparable to the LeakyReLU activation on test data.

These test results are fascinating and display the power of this novel activation scheme. Let us first examine the error plot of the the LeakyReLU network. We see, as expected, that the error increases with range – this is expected given that the norm of the received signal decreases with range, while the artificial noise added to the signal stays constant. Thus, the relative size of the noise grows with range, and we would expect the error to grow proportionally. What is unexpected is the "spiky" behavior of the RMSE plot. Typically, when one sees error plots like this, it is due to a specific instance of Gaussian noise being particularly bad for the model

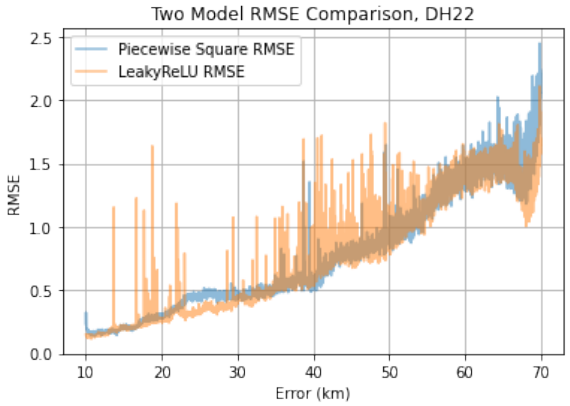

Figure 28: A comparison of the RMSE at every range between the network with LeakyReLU activation and the Piecewise Square activation. The Piecewise Square activation performs comparably, and even outperforms LeakyReLU in some respects.

under evaluation, leading to a single particularly bad prediction and a correspondingly high error. However, this cannot be the case here, since we have specifically constructed this test so that each network makes 200 predictions on distinct noisy measurements at every range, thus removing the influence of any particular instance of noise. The plot of error in the predictions made by the LeakyReLU network indicate that it is somehow inconsistent, that is, not sufficiently regularized.

The Piecewise Square network, on the other hand, does not experience these spikes to nearly the same degree. While it has noticeably higher error at certain particular ranges, it also does not experience the spikes that the LeakyReLU network does. We may conjecture that the Piecewise Square network experiences implicit regularization, likely due to its C^1 structure.

In keeping with our process of network evaluation in previous sections, we would also like to "zoom in" on a specific range and examine the outputs of the networks and compare both to the MLE at a range of 50km. These results are shown in Figure 29. We see once again that the performance of this novel architecture is similar to both a standard LeakyReLU architecture and the Maximum

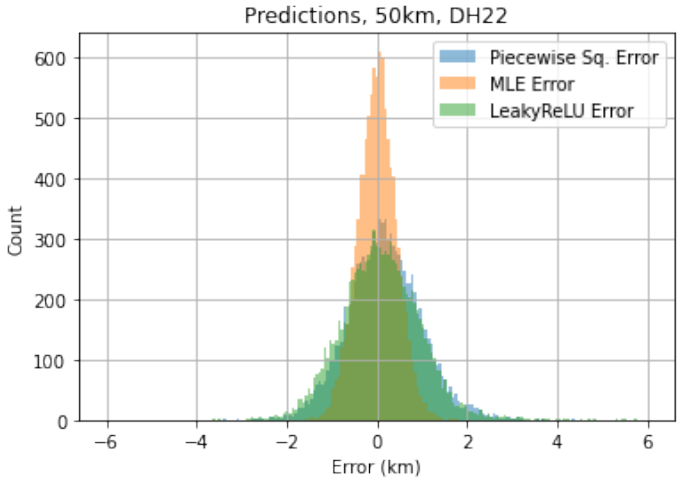

Figure 29: Histogram of the performance comparison between the MLE, a network with LeakyReLU activation, and a network with Piecewise Square activation. As we can see, the Piecewise Square activation performs comparably to the LeakyReLU activation for this problem.

Likelihood Estimator.

6.1.1 Large Test with Piecewise Square Activation – Many Duct Heights

We are also able to replicate the large test we performed with LeakyReLU in this setting. We train a neural network with the Piecewise Square activation on the same set of 15 random duct heights that we used for the experiment in Section 5.2.3. We find that the activation function is able to handle this case just as well as LeakyReLU – the RMSE for this experiment is plotted in Figure 30. The network is trained on the same set of randomly selected duct heights and tested on the same grid of duct height, receiver height, and range to give a complete categorization of the RMSE across all parameters.

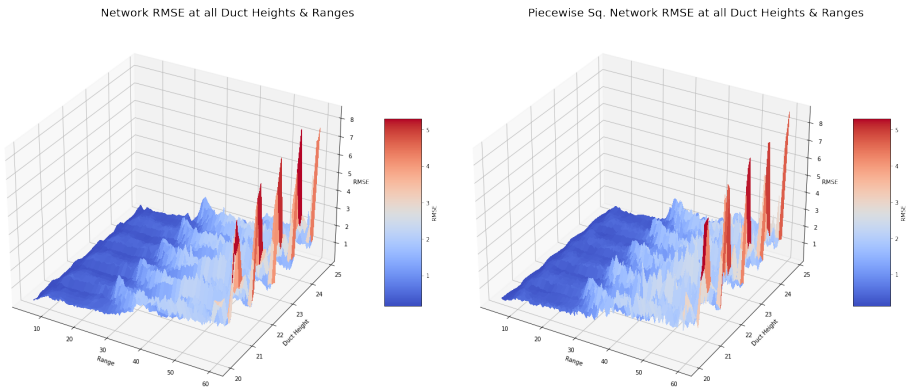

Figure 30: RMSE when a network with Piecewise Square activation is trained on 15 randomly selected duct heights between 20 and 25m. We see that this plot is nearly identical to the RMSE plot for the same experiment with the LeakyReLU activation.

6.2 Polynomial Generation for Networks in Electromagnetics

As we did for the case of applying the piecewise square architecture on the MNIST data set, we may also apply Algorithm 1 to the networks in this setting. We must make note of the differences between applying the algorithm in this case compared to the MNIST case. First, the output of the neural network in this case is one-dimensional as opposed to 10, so for a given index of the input vector and a given sample signal, there will exist a single associated polynomial. In order to show multiple examples of generated polynomials, we will compute polynomials for two given signal input vectors at 5 of the 160 possible input elements for each vector. The precise indices for which the polynomials are computed are listed in Table 12. Secondly, note that the network architecture is deeper in this network compared to the network we used to predict MNIST earlier. Therefore, it is likely that the polynomials will have a higher power here than in previous experiments.

The precise noisy measurements along with their associated noiseless electric fields are shown in Figure 31. Both are generated with the same base electric field measurement taken at 50km; noise is added to both signals, and both are normal-

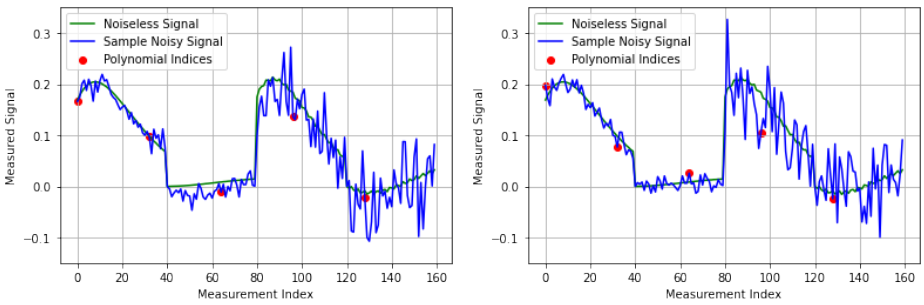

Figure 31: Two noisy signals for which we compute associated polynomials using our polynomial generation algorithm, Algorithm 1. The indices marked with red dots are the ones for which we will compute the associated polynomials.

ized separately in their horizontal and vertically polarized components. The polynomials we will generate will be the ones associated with the indices indicated in Figure 31, which are indices 0, 32, 64, 96, and 128.

The outputs of Algorithm 1 as applied to the signals given in Figure 31 are given in Table 12.

As before, we may also plot the orders of magnitude of the coefficients of these polynomials, as shown in Figure 32. We notice that unlike the MNIST example, the coefficients of polynomials on this data set do not decrease log-linearly with order. Rather, there is a particular pattern which develops in the higher order terms. We will explore this behavior further in the following section.

6.3 Analysis of Network Structure Based on Polynomial Generation

This simple list of polynomials at certain indices given in Table 12 already raises fascinating questions. For instance, examine the polynomial corresponding to signal 2, element 32. We see that the magnitude of the coefficients of this polynomial are much larger than the magnitude of other polynomials. Is this a coincidence,

Signal	Signal Element	Polynomial
Signal 1	Element 0	$51.345 - 5.094x_0 - 1.538x_0^2 - 0.329x_0^3 + 0.825x_0^4 - 0.354x_0^5 + 0.081x_0^6 - 0.01x_0^7 + 0.001x_0^8$
Signal 1	Element 32	$49.512 + 9.108x_{32} + 5.5x_{32}^2 + 2.442x_{32}^3 + 6.673x_{32}^4 + 101.424x_{32}^5 + 188.562x_{32}^6 + 132.97x_{32}^7 + 33.256x_{32}^8$
Signal 1	Element 64	$50.459 + 0.34x_{64} - 2.767x_{64}^2 - 0.184x_{64}^3 + 1.277x_{64}^4 + 0.92x_{64}^5 + 0.268x_{64}^6 + 0.039x_{64}^7 + 0.002x_{64}^8$
Signal 1	Element 96	$50.68 - 1.808x_{96} + 0.89x_{96}^2 + 1.555x_{96}^3 + 1.774x_{96}^4 + 3.769x_{96}^5 + 3.018x_{96}^6 + 1.007x_{96}^7 + 0.122x_{96}^8$
Signal 1	Element 128	$50.456 + 0.066x_{128} - 0.393x_{128}^2 + 0.131x_{128}^3 + 0.01x_{128}^4 + 0.002x_{128}^5 + 0.002x_{128}^6 + 0.001x_{128}^7$
Signal 2	Element 0	$50.843 - 5.25x_0 - 7.929x_0^2 + 3.64x_0^3 + 0.558x_0^4 + 0.339x_0^5 + 0.654x_0^6 - 0.732x_0^7 - 0.069x_0^8 + 0.279x_0^9 - 0.097x_0^{10} - 0.005x_0^{11} + 0.003x_0^{12} + 0.001x_0^{13}$
Signal 2	Element 32	$48.397 + 13.519x_{32} + 24.509x_{32}^2 - 128.614x_{32}^3 + 111.335x_{32}^4 + 39.883x_{32}^5 - 323.16x_{32}^6 + 1509.053x_{32}^7 + 5639.903x_{32}^8 + 3802.485x_{32}^9 - 26573.414x_{32}^{10} - 74318.125x_{32}^{11} - 87330.609x_{32}^{12} - 49702.676x_{32}^{13} - 19933.791x_{32}^{14} - 4348.181x_{32}^{15} - 653.218x_{32}^{16}$
Signal 2	Element 64	$49.559 - 1.218x_{64} - 1.13x_{64}^2 - 0.087x_{64}^3 + 0.089x_{64}^4 - 0.35x_{64}^5 - 0.323x_{64}^6 + 0.109x_{64}^7 + 0.144x_{64}^8 + 0.031x_{64}^9 - 0.011x_{64}^{10} - 0.01x_{64}^{11} - 0.004x_{64}^{12} - 0.001x_{64}^{13}$
Signal 2	Element 96	$50.428 - 8.823x_{96} + 0.766x_{96}^2 + 19.896x_{96}^3 + 8.752x_{96}^4 + 3.074x_{96}^5 + 18.652x_{96}^6 + 46.048x_{96}^7 + 24.722x_{96}^8 - 75.657x_{96}^9 - 156.685x_{96}^{10} - 144.101x_{96}^{11} - 80.659x_{96}^{12} - 29.705x_{96}^{13} - 7.199x_{96}^{14} - 1.073x_{96}^{15} - 0.078x_{96}^{16}$
Signal 2	Element 128	$49.5 - 1.133x_{128} - 1.96x_{128}^2 + 0.301x_{128}^3 + 0.445x_{128}^4 + 0.016x_{128}^5 + 0.045x_{128}^6 + 0.012x_{128}^7 - 0.005x_{128}^8 - 0.005x_{128}^9 - 0.003x_{128}^{10} - 0.001x_{128}^{11}$

Table 12: Select polynomials discovered by the Piecewise Polynomial network at the data points given in Figure 31.

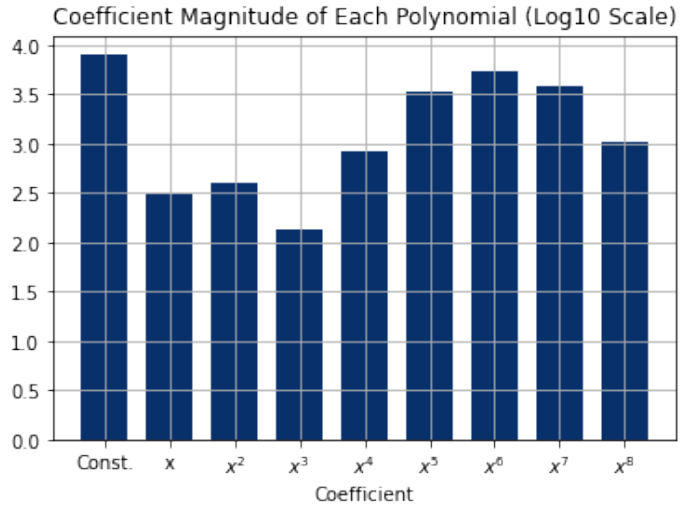

Figure 32: Plot of the log of the sums of the coefficients of the 160 polynomials associated with the left-hand signal in Figure 31. The network makes far more use of high-order terms than the network associated with MNIST.

or is there something more at work? In this section, we give an example of the type of analysis that can be done with this construction; this is not an exhaustive examination of the concepts, but the exploration of what can be done with easily computed gradients specifically is an interesting route to travel down.

We can begin to answer this question by considering what it means for a piecewise polynomial function to have large polynomial coefficients at a certain data point. As we've seen in previous examples and again in this one, the network tends to put its current prediction in the constant term of its current polynomial. Recall from Section 3.3 that the constant term in the polynomials we show is actually the sum of all the terms that do not contain the current vector element, which is very many terms. The primary effect of large coefficients in the other terms is to increase the partial derivative of the function discovered by the network in the local neighborhood of the data point under consideration. In some sense, the constant term of the polynomial is how the network makes its current prediction, while the higher order terms tell us what it uses to distinguish successive points

from one another.

In this example, what this means is that the network has discovered that element 32 is useful to distinguish measurements at a range of 50km from measurements at a range of, say, 50.1km. We can demonstrate this empirically using local partial derivatives with respect to each element of the input data. By using the polynomials associated with each element of the input data vector and the elementary Power Rule, we are able to compute the local partial derivatives of the overall predictive function given by the neural network. By concatenating the 160 partial derivatives into a single vector, we are able to form the partial derivatives into a full gradient of the function. We can use this gradient to determine which elements of the input data are useful in making local distinctions in range predictions.

We must make a note on the use of the word "gradient" here. Note that in the realm of deep learning, usually what one means by this word is the derivative of the Loss function with respect to the network parameters – that is not what we mean here. In this context, where we already have a function $f(\theta, x)$ represented by a trained network with parameters θ and data x, what we mean by gradient is the gradient of the *function* with respect to the *data*. To put it another way, we do *not* mean

$$\nabla_\theta \text{Loss} = \nabla_\theta \sum (y_i - f(\theta, x_i))^2,$$

but rather

$$\nabla_x f(\theta, x).$$

With that said, after computing all 160 partial derivatives with respect to x of $f(\theta, x)$ using the polynomials such as the ones found in Table 12, we form them into a full gradient as shown in Figure 33. We see that the large coefficients found in polynomial 32 of sample 2 are not a coincidence, but in fact, it is precisely this

portion of the input vector that the neural network uses for local information when making range predictions. Recall that the first 40 elements of the input vector correspond to the real part of the horizontal polarization of the signal; we can see in Figure 33 that this is this portion of the signal that the network uses for local information for signals at 50km.

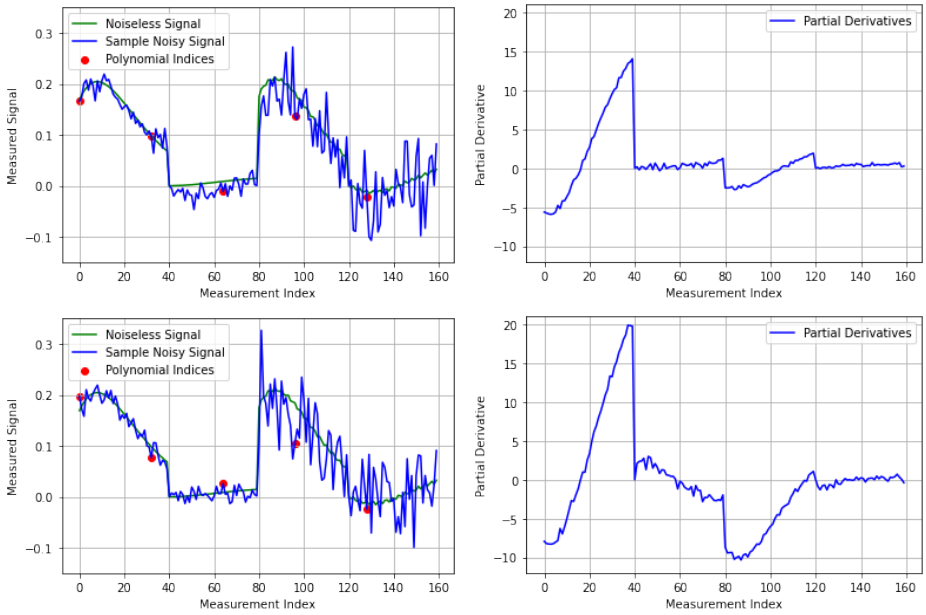

Figure 33: Left: The same two sample signals at 50km. Right: the partial derivatives of $f(\theta, x)$ at those two respective points. At a range of 50km, the real part of the horizontal polarization is the most useful part of the signal for distinguishing successive measurements, since that is the part of the signal with the highest partial derivative.

This raises yet another question: is this always the case? Does the network always use the real part of the horizontal polarization to make local distinctions in the range?

Using our novel activation and the polynomials generated by Algorithm 1, these questions are easy to answer. We may simply compute the partial derivatives for data points corresponding to other ranges and see which parts of the signal vector the network is using. In Figure 34, we perform the same computation,

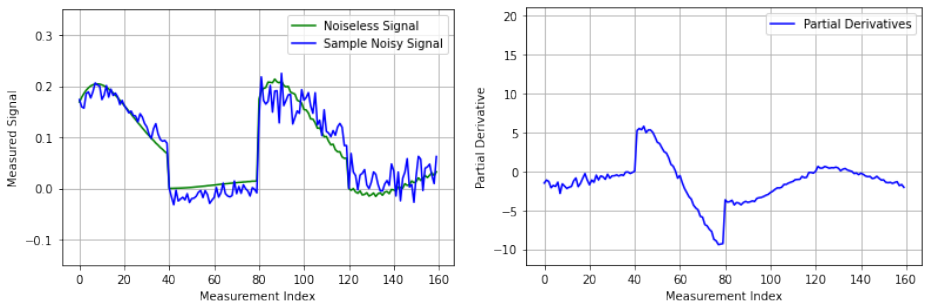

Figure 34: Left: another sample signal at a range of 30km. Right: the partial derivatives of $f(\theta, x)$ at this measurement. At a range of 30km, the imaginary part of the horizontal polarization is the most useful part of the signal for distinguishing successive measurements. This shows how local information moves to different parts of the vector as the range changes.

but with a signal corresponding to a range of 30km.

We see in Figure 34 different behavior than we saw in previous cases – the network no longer uses the first 40 elements of the input vector, corresponding to the real part of the horizontally polarized signal, to make local distinctions in range. Rather, it uses the next 40 elements, corresponding to the imaginary part of the horizontal polarization. In this way, we have discovered something about the flow of local information through the input vector – in some regions, local predictive information is in the real part of the horizontal polarization of the input, and in other areas, local information is in the imaginary part of the horizontal polarization.

We may carry this analysis even further: is this a general trend, or are these two unique cases where the sizes of the different pieces of the gradient are particularly useful? We perform one last piece of analysis, and that is to plot the norms of the different pieces of the gradient as the range of the prediction changes. We want to see which pieces of the data the network uses for local information at different ranges.

To determine this, we compute the gradient as before, but instead of consider-

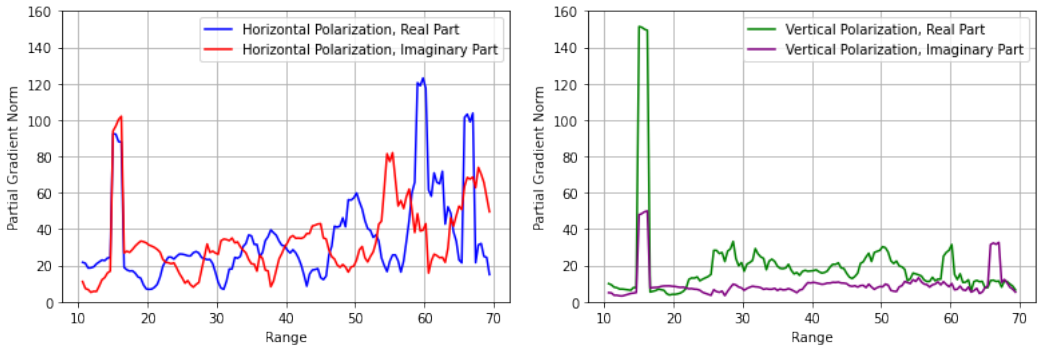

Figure 35: Left: the vector norms of the parts of the gradient of the function $f(\theta, x)$ corresponding to the real and imaginary parts of the horizontally polarized parts of the signal. Right: the vector norms of the part of the gradient corresponding to the real and imaginary parts of the vertical polarization. In the first image, we see that the network alternates between the real and imaginary parts in determining local prediction information. In the second image, the network uses the real part of the signal more throughout almost all ranges.

ing one specific range, we compute the gradient every 400m across our entire range space from 10 to 70km. We compute the norms of the four pieces of the gradient separately: the real and imaginary parts of the horizontal polarization and the real and imaginary parts of the vertical polarization. In the vectors of partial derivatives in Figures 33 and 34, these pieces correspond to elements 0 to 39, 40 to 79, 80 to 119, and 120 to 159. We plot the four curves with their corresponding ranges in Figure 35 – for ease of viewing, we plot the rolling average of the gradient norms to make the trends clear.

The results in Figure 35 are striking, especially the first image displaying the norms of the parts of the gradient corresponding to the horizontal polarization. We see that, apart from two moments of gradient blow-up at 15km and 65km, there develops a distinct pattern of alternation between norms of the real and imaginary parts. Every 8 or so kilometers, the real and imaginary parts switch places, with the network choosing to use the real part for local predictions at some intervals and the imaginary part in the following interval. We see that the two samples

we examined earlier at 30 and 50km are part of this overall trend – at 30km, the imaginary part has the larger gradient norm, while at 50km, the real part has the larger gradient norm.

These results are simultaneously exactly what we would have predicted to find, and wildly remarkable in their implications. Let us think back to what the data looks like in images such as Figure 10. We see that the data is oscillatory – imagining a horizontal slice of the data, one could easily imagine that the underlying function which would generate the data is sinusoidal. Based on Figure 35, it seems that the neural network is exactly replicating this sinusoidal structure. We did not tell the neural network to do this, and yet we see clearly that that's what the network is doing. Additionally, imagine we didn't have an image of the data to reference – finding that it has a sinusoidal structure would be remarkably insightful. Amazingly, simply writing down polynomials associated with input elements led us down a path that gave us insights about the *data itself*. Developing insight about the data manifold by the behavior of the neural network using it as input could be a powerful idea that merits further investigation.

6.3.1 Data Ambiguity with Polynomial Information

The analysis here seems likely to be linked to the notion of data ambiguity as described in Section 5.1. We have seen in previous analysis and experiments that both the neural network and Maximum Likelihood Estimator can confuse data points that are precisely 16km apart – this happens to correspond to one "cycle" of the gradient norms as shown in Figure 35. It could very well be the case that this cyclical structure in the network gradient corresponds to a similar cyclical structure in the underlying data manifold. We give here an example of what these points look

Noise Level	True Range (km)	Prediction (km)
1.0x	50.0	49.0
1.3x	50.0	54.9
2.0x	50.0	67.8
1.0x	67.8	64.9

Table 13: Predictions for the network at various noise levels. We see that, as in Section 5.1, the network makes ambiguity errors of roughly 16km when the noise is increased by 2x. In between 1x and 2x noise, we see the transitional period as the network switches between its correct and incorrect prediction.

like, and how the network behaves when it confuses specific points.

We perform one final experiment to see how the network behaves in this case of data ambiguity. We may essentially repeat the examples from Section 5.1 – we take an example signal, and increase the noise until the network confuses the given signal with another range. To that end, we start with the example signal we are familiar with at this point, associated with a range of 50km. We increase the noise on the signal, starting with the level of noise we originally trained on, and increasing until the level of noise is doubled. We see in Table 13 the predictions for the various noise levels.

We may analyze the behavior of the network more deeply in Figure 36. The first example is identical to Signal 2 in Figure 33, where local predictive information is contained in the real part of the horizontal polarization of the signal. Most interesting is the similarity between the signals at 50km and 2x noise, which is the third signal, and 67km at standard noise, the fourth signal. It becomes almost obvious how the network can confuse the two signals – the horizontal polarization, the left half of the signal, looks nearly identical, while the vertical polarization appears to have a similarly large level of noise.

The only other aspect to point out is the second set of images in Figure 36. Here, the noise is large enough that the network no longer associates the signal with a

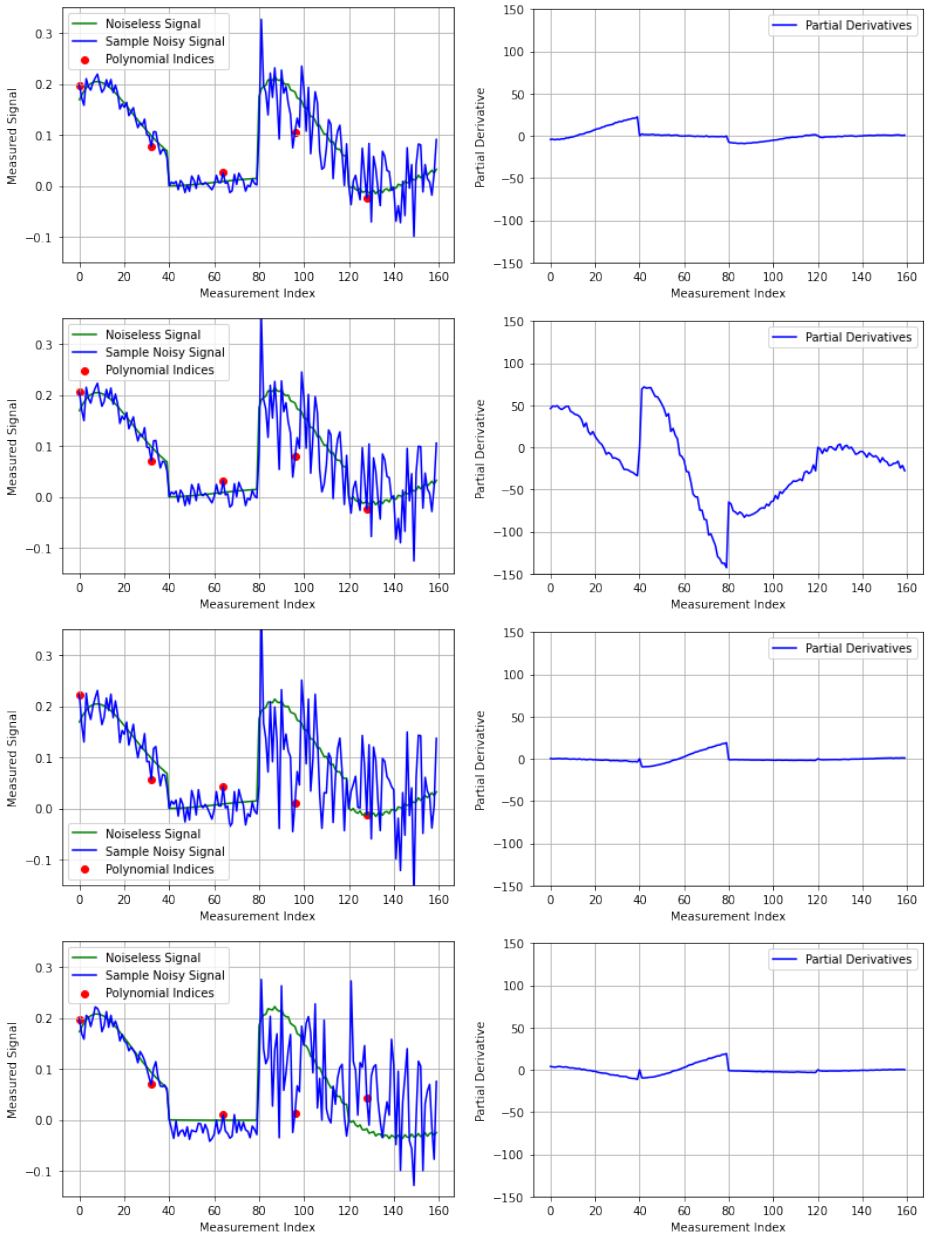

Figure 36: Rows 1 to 3: a signal at 50km with original noise level, 1.3 times the original noise level, and double the original noise level. For row 3, the network predicts the signal to be at a range of 67.8km. Row 4: the signal associated with 67.8km, with original noise level. Note the scale on the gradient plots – the gradient in the "transitional" row of 1.3 times original noise has an enormous gradient as the network shifts its prediction rapidly from 50 to 67.

range of 50km, but not so large that the network will associate the signal with 67km. Since a signal of this form is unlike all the signals the network saw in training, it does not use the region of 160-dimensional space that this signal lives in for predictions, rather, it uses this region as a transition between 50km and 67km predictions. We may confirm that this is the case when considering the gradient at this signal. As we can see in row 2 of Figure 36, the gradient at this point is enormous compared to the other points. The corresponds to the rapid transition between two similar signals associated with very different ranges – since the network is C^1, it must make a continuous transition between different predictions. When the ranges are separated by 17km and the signals differ only by a small change in the noise, it is simply necessary to have a large gradient in that region.

6.4 Remarks on Piecewise Square Activation

We've seen in this section the power of the piecewise square activation function. Not only is it comparable in performance to standard activation functions, but we've seen in a very in-depth fashion that it allows for robust analysis of the structure of the function found by the network during the training process. In some sense, it allows us to dig deep into the precise way in which the network interacts with the data. The analysis here all stems from the piecewise polynomial structure of this activation function – being able to explicitly write down the local structure of the function found by the network is remarkably powerful, and allows one to generate insight that would have been very difficult to find with other network architectures. By simply writing down a list of polynomials corresponding to input indices, numerous questions were generated that allow us to determine the local structure of the neural network function.

Let us consider a final time what we've discovered through this analysis in the context of what we already know about our data. We know upon first inspection of the data in images such as Figure 10 that the normalized data is in some sense cyclical – if we imagine trying to fit a function to a horizontal slice in the image of the data, we would probably guess that we should try to fit it with some sort of sine or cosine. If we were to fit a sine or cosine function to the data and then plot that function's derivative, we would again be left with another sinusoidal function. In some way, it makes perfect sense that after performing this sort of analysis, we would find that the network is fitting a cyclical gradient to a cyclical function. Additionally, it is likely that the bare fact of a neural network correctly finding a function which fits the underlying structure of the data would be true regardless of the activation function we use.

The remarkable thing here, then, is not *what* the neural network is finding as it fits a function to the data. The remarkable thing is that *the piecewise square activation naturally provides us a framework by which to ask what the network is doing in the first place.* We started this whole process simply by writing down polynomials, and this led us to determine the entire local structure of the network's underlying function. The advantage that using an analyzable activation function such as the piecewise square polynomial brings is precisely this insight into the resulting function, and this is part of what we should consider when determining the effectiveness of a particular activation.

7 Conclusion

In this work, we have developed a theoretical framework by which to analyze neural networks with polynomial and piecewise polynomial activation functions. Networks of polynomial activations have shown to be deconstructable into products of the form $f(\theta, x) = P(\theta)\Phi(x)$, providing a framework by which to prove a Universal Approximation Theorem for such networks. Additionally, we propose a new Piecewise Square activation which takes advantage of the relatively simple structure of polynomials to construct networks which are more analyzable than most other networks with other activation functions.

Using this and other activation functions, we have developed an extensive set of methodologies and results for performing range estimation in signal processing, including the correct choices of data inclusion, preprocessing, and network architectures. By combining all these, we are able to construct and train networks that perform comparably to the optimal model for range prediction on simulated data, which is the Maximum Likelihood Estimator.

Additionally, through the analysis brought on by our specific choice of novel activation function, we have discovered that one can uncover information about the underlying structure of the data manifold. One could imagine extending this work to any number of cases of less regular data. Using the gradient of a neural network function to uncover local information about the underlying data set is a powerful idea, and one that could be used in further explorations of this work.

8 Future Work

The further development of these items could be considered as extensions of the present work:

- Further development of a proof of a Universal Approximation Theorem for polynomial networks. A proof that for any given depth d, there exist m_1, $\ldots$, m_d large enough that the mapping given in Section 2.5 is dense in $\mathbb{R}^{2^{d-1}}$ would be sufficient as a proof, since one could combine this result with the Stone-Weierstrauss Theorem to show that networks of this form are universal approximators.

- The models for making range predictions on electric field measurements could be improved by extending them to account for the cases of the stacked ducts as described in Section 4.3. Our current models allow two parameters to vary, namely, the evaporation duct height and the receiver height. Stacked ducts themselves require three parameters to describe, so the manifold of input data goes from 2 parameters to 5, greatly increasing the manifold complexity.

- Developing a time-variant algorithm to resolve data ambiguity as described in Section 5.1. A robust way of removing false predictions and making use of true ones would greatly increase both the effective range and reliability of the prediction algorithms.

- It would be interesting to see further examples of piecewise square activation applied to other problems. As we've seen in Section 6.3, this activation provides a way to analyze a neural network and its underlying function, especially to determine how the network uses its input locally. We would like

to see this analysis extended to a problem domain where the input data is less regular. Given the deep connections between gradients of functions on a manifold and a local parameterization of a manifold, it would be interesting to see what information can be gained about a data manifold by considering the gradient of a function applied on it.

A Appendix

A.1 Network Decomposition with Piecewise Square Activation

We would like to decompose a network with Piecewise Square activation in a similar manner to the network of square activation as in Theorem 2.2. Recall that the Piecewise Square activation is given by

$$\text{Piecewise Square Activation} = \begin{cases} 0.01x - .000025 & x \leq .005 \\ x^2 & .005 \leq x \leq 1 \\ 2x - 1 & 1 \leq x. \end{cases} \tag{17}$$

Rather than layering piecewise functions over and over, we refer to the elements of $\sigma(W_i x)$ corresponding to the left, middle, and right side of the activation as $\mathbb{1}_{l_i}$, $\mathbb{1}_{m_i}$, and $\mathbb{1}_{l_i}$. We borrow the notation of indicator functions for these $\mathbb{1}$ vectors, since they are binary vectors with a 1 in the jth position if $\sigma(W_i x) < .000025$, $.000025 < \sigma(W_i x) < 1$, or $1 < \sigma(W_i x)$, respectively, and a 0 otherwise. Thus, when we take a Hadamard product of one of these vectors with $\sigma(W_i x)$, we exactly select the elements correspond to the associated part of the activation function.

We begin by analyzing $\sigma(W_1 x)$; we denote $.000025$ by a in the following.

$$\begin{aligned} \sigma(W_1 x) = &\ (.01 W_1 x - a) \odot \mathbb{1}_{l_1} + (W_1 x) \odot (W_1 x) \odot \mathbb{1}_{m_1} + (2W_1 x - 1) \odot \mathbb{1}_{r_1} \\ = &\ (.01 W_1 x - a) \odot \mathbb{1}_{l_1} + (W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} + (2W_1 x - 1) \odot \mathbb{1}_{r_1} \end{aligned}$$

We compose this first layer with the second:

$$
\begin{aligned}
\sigma(W_2\sigma(W_1x)) &= (.01W_2\sigma(W_1x) - a) \odot \mathbb{1}_{l_2} \\
&+ ((W_2\sigma(W_1x)) \odot (W_2\sigma(W_1x))) \odot \mathbb{1}_{m_2} \\
&+ (2W_2\sigma(W_1x) - 1) \odot \mathbb{1}_{r_2} \\
&= (.01W_2((.01W_1x - a) \odot \mathbb{1}_{l_1}) - a) \odot \mathbb{1}_{l_2} \\
&+ (.01W_2((W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1}) - a) \odot \mathbb{1}_{l_2} \\
&+ (.01W_2((2W_1x - 1) \odot \mathbb{1}_{r_1}) - a) \odot \mathbb{1}_{l_2} \\
&+ ((W_2\sigma(W_1x)) \odot (W_2\sigma(W_1x))) \odot \mathbb{1}_{m_2} \\
&+ (2W_2((.01W_1x - a) \odot \mathbb{1}_{l_1}) - 1) \odot \mathbb{1}_{l_2} \\
&+ (2W_2((W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1}) - 1) \odot \mathbb{1}_{l_2} \\
&+ (2W_2((2W_1x - 1) \odot \mathbb{1}_{r_1}) - 1) \odot \mathbb{1}_{l_2} \\
&= .01^2W_2(W_1x \odot \mathbb{1}_{l_1}) \odot \mathbb{1}_{l_2} - .01a(W_2\mathbb{1}_{l_1}) \odot \mathbb{1}_{l_2} - a\mathbb{1}_{l_2} \\
&+ .01W_2(W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \odot \mathbb{1}_{l_2} - a\mathbb{1}_{l_2} \\
&+ .01 \cdot 2W_2(W_1x \odot \mathbb{1}_{r_1}) \odot \mathbb{1}_{l_2} - .01(W_2\mathbb{1}_{r_1}) \odot \mathbb{1}_{l_2} - a\mathbb{1}_{l_2} \\
&+ ((W_2\sigma(W_1x)) \odot (W_2\sigma(W_1x))) \odot \mathbb{1}_{m_2} \\
&+ .01 \cdot 2W_2(W_1x \odot \mathbb{1}_{l_1}) - a(W_2\mathbb{1}_{l_1}) \odot \mathbb{1}_{r_2} - \mathbb{1}_{l_2} \\
&+ 2W_2(W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \odot \mathbb{1}_{r_2} - \mathbb{1}_{r_2} \\
&+ 2^2W_2(W_1x \odot \mathbb{1}_{r_1}) \odot \mathbb{1}_{r_2} - 2(W_2\mathbb{1}_{r_1}) \odot \mathbb{1}_{r_2} - \mathbb{1}_{r_2}
\end{aligned}
\tag{18}
$$

We handle the central product term, $((W_2\sigma(W_1x))\odot(W_2\sigma(W_1x)))\odot\mathbb{1}_{m_2}$, separately:

$$
\begin{aligned}
& ((W_2\sigma(W_1x))\odot(W_2\sigma(W_1x)))\odot\mathbb{1}_{m_2} \\
= \;& ((W_2((.01W_1x-a)\odot\mathbb{1}_{l_1}+(W_1\bullet W_1)(x\otimes x)\odot\mathbb{1}_{m_1}+(2W_1x-1)\odot\mathbb{1}_{r_1})) \\
& \odot(W_2((.01W_1x-a)\odot\mathbb{1}_{l_1}+(W_1\bullet W_1)(x\otimes x)\odot\mathbb{1}_{m_1}+(2W_1x-1)\odot\mathbb{1}_{r_1})))\odot\mathbb{1}_{m_2} \\
= \;& W_2(A+B+C)\odot W_2(A+B+C)\odot\mathbb{1}_{m_2} \\
= \;& (W_2\bullet W_2)((A+B+C)\otimes(A+B+C))\odot\mathbb{1}_{m_2} \\
= \;& (W_2\bullet W_2)((A\otimes A)+(A\otimes B)+(A\otimes C)+(B\otimes A)+ \\
& +(B\otimes B)+(B\otimes C)+(C\otimes A)+(C\otimes B)+(C\otimes C))\odot\mathbb{1}_{m_2}
\end{aligned}
$$

A, C have the form $(bW_1x-c)\odot\mathbb{1}$; when one takes a Kronecker product with these two matrices as terms, the product takes the following form. Note that we use the notation of $\mathbb{1}$ without a subscript as the identity element of the Hadamard product (i.e., a vector of all 1's) in order to decompose every term as a a Hadamard product of Kronecker products.

$$
\begin{aligned}
& (b_1W_1x-c_1)\odot\mathbb{1}_1\otimes(b_2W_1x-c_2)\odot\mathbb{1}_2 \\
= \;& (b_1W_1x\odot\mathbb{1}_1-c_1\mathbb{1}_1)\otimes(b_2W_1x\odot\mathbb{1}_2-c_2\mathbb{1}_2) \\
= \;& b_1W_1x\odot\mathbb{1}_1\otimes b_2W_1x\odot\mathbb{1}_2-b_1W_1x\odot\mathbb{1}_1\otimes c_2\mathbb{1}_2-c_1\mathbb{1}_1\otimes b_2W_1x\odot\mathbb{1}_2 \\
& +c_1\mathbb{1}_1\otimes c_2\mathbb{1}_2 \\
= \;& b_1b_2(W_1x\otimes W_1x)\odot(\mathbb{1}_1\otimes\mathbb{1}_2)-b_1c_2(W_1x\otimes\mathbb{1})\odot(\mathbb{1}_1\otimes\mathbb{1}_2) \\
& -b_2c_1(\mathbb{1}\otimes W_1x)\odot(\mathbb{1}_2\otimes\mathbb{1}_1)+c_2c_2(\mathbb{1}_1\otimes\mathbb{1}_2) \\
= \;& b_1b_2(W_1\otimes W_1)(x\otimes x)\odot(\mathbb{1}_1\otimes\mathbb{1}_2)-b_1c_2(W_1x\otimes\mathbb{1})\odot(\mathbb{1}_1\otimes\mathbb{1}_2) \\
& -b_2c_1(\mathbb{1}\otimes W_1x)\odot(\mathbb{1}_2\otimes\mathbb{1}_1)+c_2c_2(\mathbb{1}_1\otimes\mathbb{1}_2)
\end{aligned}
$$

Products of the form $B \otimes C$ or $B \otimes A$ take the form

$$\begin{aligned}
& (W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \otimes (b_2 W_1 x - c_2) \odot \mathbb{1}_2 \\
= \; & (W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \otimes b_2 W_1 x \odot \mathbb{1}_2 \\
& -(W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \otimes c_2 \mathbb{1}_2 \\
= \; & b_2(((W_1 \bullet W_1)(x \otimes x)) \otimes W_1 x) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_2) \\
& -c_2((W_1 \bullet W_1)(x \otimes x) \otimes \mathbb{1}) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_2) \\
= \; & b_2(((W_1 \bullet W_1) \otimes W_1)(x \otimes x \otimes x)) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_2) \\
& -c_2((W_1 \bullet W_1)(x \otimes x) \otimes \mathbb{1}) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_2)
\end{aligned}$$

Likewise, the reverse takes the form

$$\begin{aligned}
= \; & b_1((W_1 \otimes (W_1 \bullet W_1))(x \otimes x \otimes x)) \odot (\mathbb{1}_1 \otimes \mathbb{1}_{m_1}) \\
& -c_1(\mathbb{1} \otimes (W_1 \bullet W_1)(x \otimes x)) \odot (\mathbb{1}_1 \otimes \mathbb{1}_{m_1})
\end{aligned}$$

Finally, $B \otimes B$ is the product most familiar to us from Section 2.2; it takes the form

$$\begin{aligned}
& (W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \otimes (W_1 \bullet W_1)(x \otimes x) \odot \mathbb{1}_{m_1} \\
= \; & ((W_1 \bullet W_1)(x \otimes x) \otimes (W_1 \bullet W_1)(x \otimes x)) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_{m_2}) \\
= \; & ((W_1 \bullet W_1) \otimes (W_1 \bullet W_1))(x \otimes x \otimes x \otimes x) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_{m_1})
\end{aligned}$$

Therefore, we are left with the final expression

$$
\begin{aligned}
& ((W_2 \bullet W_2) \\
(A \otimes A) \quad & (.01^2 (W_1 \otimes W_1)(x \otimes x) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{l_1}) - .01a(W_1 x \otimes \mathbb{1}) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{l_1}) \\
& -.01a(\mathbb{1} \otimes W_1 x) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{l_1}) + a^2 (\mathbb{1}_{l_1} \otimes \mathbb{1}_{l_1})) \\
(A \otimes B) \quad & +.01((W_1 \otimes (W_1 \bullet W_1))(x \otimes x \otimes x)) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{m_1}) \\
& -a(\mathbb{1} \otimes (W_1 \bullet W_1)(x \otimes x)) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{m_1}) \\
(A \otimes C) \quad & +.01 \cdot 2(W_1 \otimes W_1)(x \otimes x) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{r_1}) - .01 \cdot 2(W_1 x \otimes \mathbb{1}) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{r_1}) \\
& -2(\mathbb{1} \otimes W_1 x) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{l_1}) + (\mathbb{1}_{l_1} \otimes \mathbb{1}_{r_1}) \\
(B \otimes A) \quad & +.01((W_1 \otimes (W_1 \bullet W_1))(x \otimes x \otimes x)) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{m_1}) \\
& -(\mathbb{1} \otimes (W_1 \bullet W_1)(x \otimes x)) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{m_1}) \\
(B \otimes B) \quad & +((W_1 \bullet W_1) \otimes (W_1 \bullet W_1))(x \otimes x \otimes x \otimes x) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_{m_1}) \\
(B \otimes C) \quad & +2(((W_1 \bullet W_1) \otimes W_1)(x \otimes x \otimes x)) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_{r_1}) \\
& -((W_1 \bullet W_1)(x \otimes x) \otimes \mathbb{1}) \odot (\mathbb{1}_{m_1} \otimes \mathbb{1}_{r_1}) \\
(C \otimes A) \quad & +2 \cdot .01(W_1 \otimes W_1)(x \otimes x) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{l_1}) - 2a(W_1 x \otimes \mathbb{1}) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{l_1}) \\
& -.01(\mathbb{1} \otimes W_1 x) \odot (\mathbb{1}_{l_1} \otimes \mathbb{1}_{r_1}) + a^2 (\mathbb{1}_{r_1} \otimes \mathbb{1}_{l_1}) \\
(C \otimes B) \quad & +2((W_1 \otimes (W_1 \bullet W_1))(x \otimes x \otimes x)) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{m_1}) \\
& -(\mathbb{1} \otimes (W_1 \bullet W_1)(x \otimes x)) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{m_1}) \\
(C \otimes C) \quad & 2 \cdot 2(W_1 \otimes W_1)(x \otimes x) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{r_1}) - 2(W_1 x \otimes \mathbb{1}) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{r_1}) \\
& -2(\mathbb{1} \otimes W_1 x) \odot (\mathbb{1}_{r_1} \otimes \mathbb{1}_{r_1}) + (\mathbb{1}_{r_1} \otimes \mathbb{1}_{r_1}) \odot \mathbb{1}_{m_2})
\end{aligned}
$$

For the final decomposition, we combine this expression with Equation 18 to decompose the Piecewise Square network.

References

[1] ALPAYDIN, E. *Introduction to machine learning*. MIT press, 2020.

[2] ANGELOV, P. P., SOARES, E. A., JIANG, R., ARNOLD, N. I., AND ATKINSON, P. M. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 11*, 5 (2021), e1424.

[3] APICELLA, A., DONNARUMMA, F., ISGRÒ, F., AND PREVETE, R. A survey on modern trainable activation functions. *Neural Networks 138* (2021), 14–32.

[4] BAI, Y., CHEN, W., CHEN, J., AND GUO, W. Deep learning methods for solving linear inverse problems: Research directions and paradigms. *Signal Processing 177* (2020), 107729.

[5] BENGIO, Y., GOODFELLOW, I., AND COURVILLE, A. *Deep learning*, vol. 1. MIT press Cambridge, MA, USA, 2017.

[6] BERG, J., AND NYSTRÖM, K. Neural network augmented inverse problems for pdes. *arXiv preprint arXiv:1712.09685* (2017).

[7] BERTHIAUME, D., PAFFENROTH, R., AND GUO, L. Understanding deep learning: Expected spanning dimension and controlling the flexibility of neural networks. *Frontiers in Applied Mathematics and Statistics* (2020), 52.

[8] BURKART, N., AND HUBER, M. F. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research 70* (2021), 245–317.

[9] CASTELVECCHI, D. Can we open the black box of ai? *Nature News 538*, 7623 (2016), 20.

[10] CASTRO, J. L., MANTAS, C. J., AND BENITEZ, J. Neural networks with a continuous squashing function in the output are universal approximators. *Neural Networks 13*, 6 (2000), 561–563.

[11] CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems 2*, 4 (1989), 303–314.

[12] DAYHOFF, J. E., AND DELEO, J. M. Artificial neural networks: opening the black box. *Cancer: Interdisciplinary International Journal of the American Cancer Society 91*, S8 (2001), 1615–1635.

[13] DOĞANÇAY, K. Passive emitter localization using weighted instrumental variables. *Signal processing 84*, 3 (2004), 487–497.

[14] DOŠILOVIĆ, F. K., BRČIĆ, M., AND HLUPIĆ, N. Explainable artificial intelligence: A survey. In *2018 41st International convention on information and communication technology, electronics and microelectronics (MIPRO)* (2018), IEEE, pp. 0210–0215.

[15] FERGUSON, E. L., RAMAKRISHNAN, R., WILLIAMS, S. B., AND JIN, C. T. Convolutional neural networks for passive monitoring of a shallow water environment using a single sensor. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2017), IEEE, pp. 2657–2661.

[16] FERGUSON, E. L., WILLIAMS, S. B., AND JIN, C. T. Sound source localization in a multipath environment using convolutional neural networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2018), IEEE, pp. 2386–2390.

[17] FISHER, R. A. On the mathematical foundations of theoretical statistics. *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character 222*, 594-604 (1922), 309–368.

[18] GLOROT, X., BORDES, A., AND BENGIO, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (2011), JMLR Workshop and Conference Proceedings, pp. 315–323.

[19] HAMPTON, J. R. The impact of evaporative ducting on covert communications. In *MILCOM 2007-IEEE Military Communications Conference* (2007), IEEE, pp. 1–7.

[20] HENDRYCKS, D., AND GIMPEL, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415* (2016).

[21] HITNEY, H. V., RICHTER, J. H., PAPPERT, R. A., ANDERSON, K. D., AND BAUMGARTNER, G. B. Tropospheric radio propagation assessment. *Proceedings of the IEEE 73*, 2 (1985), 265–283.

[22] HOFFMANN, F., SCHILY, H., CHARLISH, A., RITCHIE, M., AND GRIFFITHS, H. A rollout based path planner for emitter localization. In *2019 22th International Conference on Information Fusion (FUSION)* (2019), IEEE, pp. 1–8.

[23] HOLZINGER, A., MALLE, B., SARANTI, A., AND PFEIFER, B. Towards multimodal causability with graph neural networks enabling information fusion for explainable ai. *Information Fusion 71* (2021), 28–37.

[24] HORNIK, K., STINCHCOMBE, M., AND WHITE, H. Multilayer feedforward networks are universal approximators. *Neural networks 2*, 5 (1989), 359–366.

[25] JARRETT, K., KAVUKCUOGLU, K., RANZATO, M., AND LECUN, Y. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th international conference on computer vision* (2009), IEEE, pp. 2146–2153.

[26] KHATRI, C., AND RAO, C. R. Solutions to some functional equations and their applications to characterization of probability distributions. *Sankhyā: The Indian Journal of Statistics, Series A* (1968), 167–180.

[27] KIDGER, P., AND LYONS, T. Universal approximation with deep narrow networks. In *Conference on learning theory* (2020), PMLR, pp. 2306–2327.

[28] KISEL'ÁK, J., LU, Y., ŠVIHRA, J., SZÉPE, P., AND STEHLÍK, M. "spocu": scaled polynomial constant unit activation function. *Neural computing and applications 33*, 8 (2021), 3385–3401.

[29] KLAMBAUER, G., UNTERTHINER, T., MAYR, A., AND HOCHREITER, S. Self-normalizing neural networks. *Advances in neural information processing systems 30* (2017).

[30] LECUN, Y., CORTES, C., AND BURGES, C. The mnist dataset of handwritten digits (images). *NYU: New York, NY, USA* (1999).

[31] LIPTON, Z. C. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue 16*, 3 (2018), 31–57.

[32] LIU, S., TRENKLER, G., ET AL. Hadamard, khatri-rao, kronecker and other matrix products. *Int. J. Inf. Syst. Sci 4*, 1 (2008), 160–177.

[33] LIU, W., YANG, Y., XU, M., LÜ, L., LIU, Z., AND SHI, Y. Source localization in the deep ocean using a convolutional neural network. *The Journal of the Acoustical Society of America 147*, 4 (2020), EL314–EL319.

[34] MAAS, A. L., HANNUN, A. Y., NG, A. Y., ET AL. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml* (2013), vol. 30, Citeseer, p. 3.

[35] MARCEL, S., AND RODRIGUEZ, Y. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia* (2010), pp. 1485–1488.

[36] NAIR, V., AND HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In *Icml* (2010).

[37] O'CONNOR, A., SETLUR, P., AND DEVROYE, N. Single-sensor rf emitter localization based on multipath exploitation. *IEEE Transactions on Aerospace and Electronic Systems 51*, 3 (2015), 1635–1651.

[38] OUARTI, N., AND CARMONA, D. Out of the black box: Properties of deep neural networks and their applications. *arXiv preprint arXiv:1808.04433* (2018).

[39] OZGUN, O., APAYDIN, G., KUZUOGLU, M., AND SEVGI, L. Petool: Matlab-based one-way and two-way split-step parabolic equation tool for radiowave propagation over variable terrain. *Computer Physics Communications 182*, 12 (2011), 2638–2654.

[40] PAKRAVAN, S., MISTANI, P. A., ARAGON-CALVO, M. A., AND GIBOU, F. Solving inverse-pde problems with physics-aware neural networks. *Journal of Computational Physics 440* (2021), 110414.

[41] PARK, J., KIM, M. J., JUNG, W., AND AHN, J. H. Aespa: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699* (2022).

[42] PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L., AND LERER, A. Automatic differentiation in pytorch.

[43] POOR, H. V. *An introduction to signal detection and estimation*. Springer Science & Business Media, 2013.

[44] RAISSI, M., PERDIKARIS, P., AND KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics 378* (2019), 686–707.

[45] ROGERS, L. T., HATTAN, C. P., AND STAPLETON, J. K. Estimating evaporation duct heights from radar sea echo. *Radio Science 35*, 4 (2000), 955–966.

[46] SCHMITZ, J., MATHAR, R., AND DORSCH, D. Compressed time difference of arrival based emitter localization. In *2015 3rd International Workshop on Compressed Sensing Theory and its Applications to Radar, Sonar and Remote Sensing (CoSeRa)* (2015), IEEE, pp. 263–267.

[47] SHAREEF, A., ZHU, Y., AND MUSAVI, M. Localization using neural networks in wireless sensor networks. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications* (2008), pp. 1–7.

[48] SLYUSAR, V. New operations of matrix products for application of radars. In *IEEE MTT/ED/AP West Ukraine Chapter DIPED-97. Direct and Inverse Problems of Electromagnetic and Acoustic Theory (IEEE Cat. No. 97TH8343)* (1997), IEEE, pp. 73–74.

[49] STANSFIELD, R. Statistical theory of df fixing. *Journal of the Institution of Electrical Engineers-Part IIIA: Radiocommunication 94*, 15 (1947), 762–770.

[50] STONE, M. H. Applications of the theory of boolean rings to general topology. *Transactions of the American Mathematical Society 41*, 3 (1937), 375–481.

[51] TIKK, D., KÓCZY, L. T., AND GEDEON, T. D. A survey on universal approximation and its limits in soft computing techniques. *International Journal of Approximate Reasoning 33*, 2 (2003), 185–202.

[52] WANG, J., CHEN, L., AND NG, C. W. W. A new class of polynomial activation functions of deep learning for precipitation forecasting. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining* (2022), pp. 1025–1035.

[53] WANG, X., WU, Y., ZHANG, A., HE, X., AND CHUA, T.-S. Towards multi-grained explainability for graph neural networks. *Advances in Neural Information Processing Systems 34* (2021), 18446–18458.

[54] WITZ, E., BARGER, M., AND PAFFENROTH, R. Deep learning for range localization via over-water electromagnetic signals. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)* (2021), IEEE, pp. 1537–1544.

[55] XU, F., USZKOREIT, H., DU, Y., FAN, W., ZHAO, D., AND ZHU, J. Explainable ai: A brief survey on history, research areas, approaches and challenges. In *CCF international conference on natural language processing and Chinese computing* (2019), Springer, pp. 563–574.

[56] YANG, Z., ZHANG, A., AND SUDJIANTO, A. Enhancing explainability of neural networks through architecture constraints. *IEEE Transactions on Neural Networks and Learning Systems 32*, 6 (2020), 2610–2621.

[57] YAROTSKY, D. Universal approximations of invariant maps by neural networks. *Constructive Approximation 55*, 1 (2022), 407–474.

[58] YUAN, H., YU, H., WANG, J., LI, K., AND JI, S. On explainability of graph neural networks via subgraph explorations. In *International Conference on Machine Learning* (2021), PMLR, pp. 12241–12252.

[59] ZHAO, W., ZHAO, J., LI, J., ZHAO, D., HUANG, L., ZHU, J., LU, J., AND WANG, X. An evaporation duct height prediction model based on a long short-term memory neural network. *IEEE Transactions on Antennas and Propagation 69*, 11 (2021), 7795–7804.

[60] ZHAO, X., YARDIM, C., WANG, D., AND HOWE, B. M. Estimating range-dependent evaporation duct height. *Journal of Atmospheric and Oceanic Technology 34*, 5 (2017), 1113–1123.

[61] ZHOU, J., QIAN, H., LU, X., DUAN, Z., HUANG, H., AND SHAO, Z. Polynomial activation neural networks: Modeling, stability analysis and coverage bp-training. *Neurocomputing 359* (2019), 227–240.

[62] ZHU, X., LI, J., ZHU, M., JIANG, Z., AND LI, Y. An evaporation duct height prediction method based on deep learning. *IEEE Geoscience and Remote Sensing Letters 15*, 9 (2018), 1307–1311.