A Deep Learning Example: the Generic Vector Projection Model, Interpolation and Generalization Analysis

Linda Ness

Abstract The simplest deep learning models for functions are compositions of non-linear activation functions applied to affine linear functions. The Universal Approximation Theorem for Neural Networks implies that any Borel measurable function can be approximated using this class of models.This powerful class of functions is proving to be remarkably useful and underpins the explosion of artificial intelligence applications. Mathematically one is seeking a representation computed by a composition of all but the final function that accurately predicts the function values by the final linear function. In this paper we mathematically analyze a simple, but general two-layer model where the first layer function represents the data by applying the biased projections of the data onto a single vector generic for the training data set.The Relu activation function ensures that the solution for the the second layer solution may easily be computed exactly by solving an upper triangular matrix. This model gives us a huge family of models which exactly interpolate the training data. We can minimize the generalization error for a sample over this family and minimize the average generalization error for a set of 'leave out k samples' . Properties of the model are proved exploiting the geometry of the model parameters and data. A number of current research questions can be answered for this simple model. The model exploits recent theoretical research shown to be effective: adaptation of the architecture to the data, positive homogeneity, parallel architectures, unbounded width, zero training error, and data dependent kernels. The paper does not assume deep learning knowledge and the proofs are self-contained.

1 Introduction

Deep learning models have proven to be very effective for efficiently learning functions on important types of real world data, for example: [19] [24], [30]. Typically,

Linda Ness
Rutgers, Piscataway, NJ, e-mail: linda.ness@rutgers.edu

models are efficiently identified by stochastic gradient in the huge space of parameters defining the models using large, but relatively smaller training sets. The most basic models are compositions of nonlinear activation functions applied to affine linear functions. There is a vast research literature. Some excellent overviews and references are given in [15] [32] [1] [2].

Research questions include:

- Which models are reachable via gradient descent? Which models have optimal optimal training error? What are geometric characterizations of their parameter loci in the parameter space (the loss landscape)?
- How does the empirical generalizability vary among optimal and reachable models? What properties improve optimization and generalization? What predictions about generalizability can be made using data and label dependent properties?
- What are the learning properties (including the learning algorithm's dynamics) of specific classes of models? What are the properties of the representation of the data computed in an architecture before the final prediction?

Examples of recent research progress include: zero training loss can be achieved without significantly negatively impacting generalization in highly over parametrized networks [33]. Two layer network architectures are being extensively studied [1] [11] [8] [3] . For example, a condition on a Relu-based kernel on the data has been identified and shown to imply convergence of gradient descent to zero training error if the network has enough hidden neurons[11]. Other properties of the kernel characterize the number of iterations needed to achieve a target accuracy [1]. The kernel together with the data labels can predict test accuracy [1]. The behavior of trained infinite width networks with randomly initialized parameters has been characterized by another kernel - the Neural Tangent Kernel [18] [2]. Positive homogeneity and parallel architectures exploiting adaptation of network architecture to data have been shown to improve optimization [16] [6]. In addition: loss landscapes for the standard loss function have been proved to be very high-dimensional submanifolds of the parameter space [9]; for two-layer autoencoder networks, the Frobenius norm loss landscape, which can be analyzed using Morse theory, reduces the symmetry group to the orthogonal transformations and results in critical points where the encoder and decoder are transposes of each other [20].

The focus of this paper is the definition, geometric explanation and mathematical analysis of a different very simple two layer architecture for a real-valued function which we call a GVP neural net function(Generic Vector Projection). The model's simplicity make it possible to compute it by hand for small examples, even though it is quite general and the simplicity of the model's geometry makes its easier to prove and understand its properties. Unlike the cited research on two-layer networks, no statistical assumptions are made on the data distribution, making it easier to reason geometrically in terms of the data. For this simple model a number of the research

questions can be answered. Additionally, analysis revealed that the model exploits principles that recent theoretical research has been shown to be effective: adaptation of the architecture to the data, positive homogeneity, parallel architectures, unbounded width, zero training error, and data dependent kernels.

1.1 Paper Overview

Section 2 defines the basic model for deep neural nets and poses two questions about the representation computed by the first layers of the model. Section 3 focuses on depth one neural networks, proving necessary conditions for functions represented by them and then using the conditions to give a simple geometric proof that the *xor* function cannot be represented using a depth one neural network. The section also defines the Generic Vector Projection (GVP) representation of a data set, proves properties, and deduces as a corollary that a one-layer neural network function defined on such a representation can exactly interpolate a function defined on a data set (zero training error). This is applied to compute a two layer GVP model for the *xor* function. Section 4 first computes the gradient descent formula for one-layer networks and proves a lower bound on the loss outside the positive affine cone determined by the data. Next gradient descent for a GVP representation of data is proved to converge to a solution of a GVP model if the initialization is in the positive affine cone determined by the representation. Section 5 defines empirical and theoretical generalizability, proves that empirical generalization error is minimized on the unit sphere (the closure of the parameter space of GVP functions) , and proves that the empirical generalization error is geodesically convex on the closure of the components of parameter space for GVP functions. This section also defines a multi-scale non-parametric representation of measures, call the product formula representation, and proposes applications to compare and summarize different training, testing and execution data sets and generalization error functions for different training and testing data sets. Section 6 reprises the GVP Model. Section 7 summarizes the paper and propose a number of experimental and research directions. The reader may find it helpful to first read the paper introduction and the reprise of the model in section 6 before reading the rest of the paper.

2 The Model

The simplest deep learning models for functions are compositions of affine transformations followed by simple functions on the matrix entries. The *depth* of the model is the number of such compositions in the model. Such a function $Y : \mathbb{R}^{n+1} \to \mathbb{R}^m$ is defined by

$$Y(X) = f \circ H^L \circ f \circ H^{L-1} \circ \ldots \circ f \circ H^1(X) \tag{1}$$

where the H^i are affine transformations and $f : \mathbb{R} \to \mathbb{R}$ is a simple function applied to each matrix entry. The superscript i denotes the level in the composition. The activation functions f are assumed to be differentiable except at a finite number of points. The most common f is the ReLu function defined by

$$f(x) = 0 \text{ if } x < 0 \tag{2}$$

$$f(x) = x \text{ if } x >= 0 \tag{3}$$

Remark 1. Throughout we will assume that the activation function f is the *ReLu* function.

Each affine transformation $H : \mathbb{R}^{n+1} \to \mathbb{R}^{m+1}$ can be represented by a matrix with $m+1$ rows and $n+1$ columns operating on the $n+1$ affine coordinates for a point $x \in \mathbb{R}^n$ (coordinate for x followed by a 1).

$$H = \begin{bmatrix} W & B \\ 0 & 1 \end{bmatrix} \tag{4}$$

Here W is an $m \times n$ matrix referred to as the weight matrix, B is an $m \times 1$ matrix referred to as the bias matrix, 0 denotes a $1 \times n$ matrix, and 1 denotes a number (i.e. a $1x1$ matrix). H computes affine coordinates. In this simple model there are no constraints among the matrix entries defining the affine transformations.

By the Universal Approximation Theorems [21] [17] [10] [15] (Section 6.4.1) [14] large classes of functions, including continuous and Borel functions functions can be approximated to arbitrarily small precision by functions in this class. Thus essentially all random variables can be approximated – a much bigger class than is typically exploited in statistics.

Given a data set, i.e. a finite set $D \subset \mathbb{R}^n$, and values of a non-negative function $\bar{Y}$ on D, the challenge is to find a model function Y which minimizes a given real-valued, differentiable loss function $L(Y, \bar{Y})$ on a training data set $D_{tr} \subset D$. The standard L^2 loss function is $||Y - \bar{Y}||^2$. The models are restricted to a particular *architecture*: the number of levels, a specification of the dimensions of the domain and range spaces of the intermediate affine functions and the pointwise activation function. This restricts the number of variables entries defining the affine transformations to be a particular (large) finite number. A minimizer is typically heuristically found by gradient descent (or stochastic gradient descent) on the variables in the affine transformation, beginning from some initialization (often random).

2.1 Representation Questions

We can view the the first $L-1$ layers of a neural net function (formula 1) as computing a representation $R(D)$ of the data.

Representation questions include:

- What are properties of a representation which guarantees that composition with the last layer function will interpolate the function exactly on a training data set $D_{tr} \subset D$ (i.e. have zero training error)?

$$f \circ H^L(R(D_{tr})) = \bar{Y}(D_{tr}) \tag{5}$$

- What geometric relationship must hold between the representations of the training and test set, $R(D_{tr})$ and $R(D_{te})$, to guarantee that the trained model Y is an optimal empirical generalizer on $R(D_{te})$?

A general observation which helps address Question 2 is that the first $L-1$ layers of the neural net function determine a function which is positive homogenous in the first $L-1$ affine functions. If the i^{th} weight and bias matrices are multiplied by a positive constant t_i, the non-affine values of the representation vectors are multiplied by their product. However, because the last layer is linear and positive homogeneous a previous solution can be divided by the same factor. Thus it suffices to analyze the function family of models Y on a product of $L-1$ spheres, one for each layer.

3 Real Depth 1 Neural Network Functions.

We begin by addressing *Question 1* in the simplest case: a real-valued function ($m = 1$) defined by a depth one model ($L = 1$), so

$$Y(x) = f \circ H(\bar{x}) \tag{6}$$

where $\bar{x}$ denotes affine coordinates for x. The matrix W in H in equation 4 is a 1 x n vector w and the matrix B in H is a number b. Equivalently using equation 4, the non-affine coordinates for Y are:

$$Y(x) = \langle w,x\rangle + b \text{ if } \langle w,x\rangle + b > 0 \tag{7}$$
$$Y(x) = 0 \text{ if } \langle w,x\rangle + b \leq 0 \tag{8}$$

In this case, Y is determined by the projection of x onto w and b,

$$proj_w(x) = \frac{< w,x >}{||w||} \frac{w}{||w||} \tag{9}$$

$Y(x) =< w, proj_w(x) > +b$, if the length of the projection translated by b is positive and zero otherwise. Thus Y is constant on hyperplanes perpendicular to w and zero on hyperplanes in the non-positive half-space $\langle w,x\rangle + b \leq 0$.

Given a finite data set $D \subset \mathbb{R}^n$, $Y(D) = Y(proj_w(D))$. This property determines necessary and sufficient criterion for Y to interpolate a function $\bar{Y} : D \to \mathbb{R}$ defined on a finite data set $D \subset \mathbb{R}^n$. Given $\bar{Y}$, for $d \in D$ define $\bar{Y}_w : proj_w(D) \to \mathbb{R}$ by

$$\bar{Y}_w(proj_w(d)) = \bar{Y}(d) \tag{10}$$

$\bar{Y}_w$ is well-defined if and only if distinct points in D project to distinct points on the line in the direction, $|proj_w(D)| = |D|$, or $|proj_w(D)| < |D|$ and $\bar{Y}$ has the same value on points with the same projection.

Lemma 1. ***Necessary Conditions for NN Interpolation*** *If a depth one real-valued neural network function $Y : \mathbb{R}^n \to \mathbb{R}$ defined by equation 4 interpolates a non-negative function $\bar{Y} : D \to \mathbb{R}$ defined on a finite data set $D \subset \mathbb{R}^n$, then*

1. *$Y(proj_w(D)) = \bar{Y}_w(proj_w(d))$*
2. *$\bar{Y}_w$ is well-defined*
3. *$\bar{Y}_w$ is monotonically non-decreasing*
4. *On the subset $P \subset D$ where $\bar{Y}$ is positive $\bar{Y}_w(proj_w(P))$ is monotonically increasing and its values are linearly interpolated on the line with direction $\frac{w}{||w||}$*

Proof. The proof follows immediately from the definition of $\bar{Y}_w$ in equation 10. □

The last condition of the lemma is a very strong necessary condition.

3.1 Application to the xor Function

The *xor* function is defined on the four data points $D = \{(0,0),(1,1),(1,0),(0,1)\}$ to have values $0,0,1,1$, respectively. Over the field $\mathbb{F}_2$ of characteristic 2, *xor* is just the sum function and hence is interpolated exactly by a linear function over $\mathbb{F}_2$.

The second and third necessary conditions in the lemma show that the *xor* cannot be interpolated exactly by a real valued neural network function of depth 1. If a line has positive slope (e.g. x = y), the *xor* values occur in the order 0, 1, 0. If a line has negative slope (x = -y), the *xor* values occur in the order 1, 0, 1. Both violating the monotonicity condition. For the axes, the induced function on the projected points is multi-valued so xor_w is not well-defined.

3.2 Implications for representations of data

Given a representation $R(D) \subset \mathbb{R}^{p+1}$ of a finite data set $D \subset \mathbb{R}^n$ (perhaps computed by a neural network with large depth), and a function $\bar{Y} : D \to \mathbb{R}^k$ defined on the data set, any choice of k vectors $w_j \in \mathbb{R}^p$ and numbers b_j determine a a potential neural network model. If this model is to interpolate each of the component functions of $\bar{Y}$, there must exist vectors w_j such that the functions $\bar{Y}_{w_j}$ for $j = 1,...k$ induced on

the lines in the directions determined by w_j are all monotonically non-decreasing and increasing on the subsets of D for which the values are positive and interpolate the function induced on the line $\bar{Y}_w$ The representation has be in a high enough dimension (or have a sufficiently informative set of lines) to re-order the data as required by each component function.[1] Independence of the representations of the training data points is sufficient but the linear equation solving for the parameters of the last layer should also be easy to solve (i.e. not require computing the inverse of a general matrix). If the representation vectors are not independent, the standard way of approximating the solution to the linear equation defining the last layer is to solve the regression problem using the normal equations [15] (Section 5.1.4). These equations in general require computing the pseudo-inverse known as the Moore Penrose inverse [29] (pp. 457-458). See [15] (Section 5.7.2) for a discussion of representation from the point of view of kernels.

3.3 The generic vector projection representation of data

Given a finite data set $D \subset \mathbb{R}^n$ of cardinality $|D| = m$, define a vector $w \in \mathbb{R}^n$ to be *generic for D* if D projects onto m distinct points on the line in the direction of w. Let $\lambda(w) = \langle w, D \rangle$. The set G_D of vectors w generic for D is the complement of a finite union of hyperplanes and thus is a union of a set of connected components, each of which is a positive affine cone. Each pair of distinct points x_i and x_j in D define a hyperplane $H_{i,j}$ in R^n

$$H_{x_i,x_j} = \{x :< x, x_i - x_j >= 0\} \tag{11}$$

These are the vectors for which x_i and x_j project onto the same vector on the line in the direction of x. The set of vectors which are not generic for w given D is the union of these hyperplanes.

$$H_D = \cup_{x_i,x_j \in D, x_i \neq x_j} H_{x_i,x_j} \tag{12}$$

Thus

$$G_D = \mathbb{R}^n - H_D \tag{13}$$

[1] Note that the function $\bar{Y}$ determines a similarity function $exp(Y(x_i) - Y(x_j))^2/\sigma)$ on the complete graph whose nodes correspond to the training data points. The training data can be represented using the values of the second eigenfunction (spectral clustering [31]). This reorders the points so that they cluster to distinguish points with different values of $\bar{Y}$ but may not be monotonic or guarantee linear interpolation. Training the neural net extends $\bar{Y}$ and hence the similarity function to the test data, enabling the representation to be extended using the Nyström method.[5] [7]

For w in each component of G_D, the graphs of the m continuous evaluation functions functions $\lambda_i(w) =< w, x_i >$, $x_i \in D$ do not cross, so we may assume the indices of the data points are ordered so that $\lambda_1 < ... < \lambda_m$ for w in a component of G_D [2].

Given a vector $w \in \mathbb{R}^n$ let W_w denote the matrix with m rows, each of whose rows is the vector w.

We define the *generic vector projection representation GVP of D determined by* $w \in G_D$ to be the one-layer neural network function

$$R_w(D) = f(H_w(D)) \tag{14}$$

where H_w is the affine linear transformation defined by the $m+1$ by $n+1$ matrix

$$H_w = \begin{bmatrix} W_w & -\lambda(w) \\ 0 & 1 \end{bmatrix} \tag{15}$$

The last row of H_w has n zeros followed by a one. Let $R_w = f \circ H_w$. We define the *generic vector projection representation of D* , $GVP_w(D)$ to be:

$$R_w(D) = f(H_w D) \tag{16}$$

where $f = Relu$. $R_w(D)$ is an affine coordinate representation of D in $\mathbb{R}^{m+1}$, so there are enough dimensions for the representations of the m points to be independent. The application of Relu forces the coordinates of the representation to be zero or positive.

Lemma 2. ***GVP representation lemma***
Assume w is a generic vector for the data set D and $|D| = m$. *Let* $\lambda_i =< w, x_i >$, $x_i \in D$, *where D is ordered so that the* λ_i*'s are (strictly) increasing. Let* $\bar{x}_i \in \mathbb{R}^{n+1}$ *denote affine coordinates for* x_i.

1. $R_w(\bar{x}_j)_i = f(\lambda_j - \lambda_i)$ *for i = 1, ..., m*
2. $R_w(\bar{x}_j)_{m+1} = 1$
3. *The first m rows of* $R_w(D)$ *form an upper triangular matrix with zeros on and below the diagonal. The above diagonal entries* ($j > i$) *are positive* $R(\bar{x}_j)_i = \lambda_j - \lambda_i$.
4. *The vector representations* $R_w(\bar{x}_j)$ *of the m data points in D are m independent vectors in* $\mathbb{R}^{m+1}$ *(in affine coordinates). The* m^{th} *unit vector* e_m *is orthogonal to all of the representation vectors* $R_w(\bar{x}_j)$, $j = 1, ..., m$.
5. *Let X denote the matrix whose first m columns are* $R(D)$ *and whose last column is* e_m. *For* $y \in \mathbb{R}^{m+1}$, *the equation* $y = cX$ *has a unique solution*
6. *The unique solution to* $y = cX$ *is* $c_{m+1} = y_1$, $c_m = y_{m+1}$ *and*

$$\frac{y_j - y_{j-1}}{\lambda_j - \lambda_{j-1}} = c_1 + ... + c_{j-1} \text{ for } j = 2, ..., m \tag{17}$$

[2] The permutations generated by pairs of component orders form a subgroup of the permutations on m elements. This subgroup provides an abstract summary of the training data which indicates how many more dimensions are needed to represent a general function

7. *The representation is positive homogenous in w: for $t > 0, R_{tw} = tR_w$ of degree* 1. *The unique solution is positive homogeneous of degree* 0.

Proof. The first statement is implied by $R_w(\bar{x}_j)_i = f(< w, x_j > -\lambda_i) = f(\lambda_j - \lambda_i)$ for i = 1, ..., m. The second statement is implied by the fact that H_w maps affine coordinates to affine coordinates. The third and fourth statements follow immediately from the first two. The fifth statement is true because the rows of X are independent. The equation in the sixth statement can be explicitly checked. Positive homogenity follows from linearity of the inner product functional:$< tw, x >= t < w, x >$. □

Corollary 1. ***Neural Net Interpolation*** *Assume $\bar{Y} : D \to \mathbb{R}^k$ is a non-negative function on a finite data set $D \subset \mathbb{R}^n$ of cardinality $|D| = m$*

1. *For any vector $w \in \mathbb{R}^n$, generic for D, there exists a one-layer neural net function $Y : \mathbb{R}^{m+1} \to \mathbb{R}^k$, $Y = f \circ c$, such that $Y \circ R_w$ interpolates $\bar{Y}$ on D.*
2. *For any vector w such that $\bar{Y}_w$ is well-defined, there exists a one-layer neural net function $Y : \mathbb{R}^{|proj_w(D)|+1} \to \mathbb{R}^k$, $Y_w = f \circ c$ such that $Y_w = Y \circ R_w$ interpolates $\bar{Y}$ on D.*
3. *Y_w depends only on the direction of w: for $t > 0$, $Y_{tw} = Y_w$, i.e. is homogenous of degree* 0.

Proof. We may assume D is ordered so that the components of $\lambda = \langle w, D \rangle = (\lambda_1, ..., \lambda_m)$ are strictly increasing. We first prove the corollary for k = 1, Denote the values of $\bar{Y}$ on D by $y = (y_1, ..., y_m)$. Part five of the lemma shows that the equation $y = cX$ has a unique solution. Since $y \geq 0$, $y = f(cX)$ [3]. If $k > 1$, compute the solution for each component. For the second claim, if $|proj_w(D)| < m$ several points in D project to the same value on w and since $\bar{Y}_w$ is well-defined the $\bar{Y}$ has the same value on these so we can de-duplicate the set D to eliminate the extra points and solve it as before. The interpolation depends only on the projection onto w so the neural net function interpolates the omitted points correctly. The last claim is true because equation 17 together with the Neural Net Interpolation Corollary 1 imply $Y_{tw} = Y_w$. □

3.4 GVP representation with separation bias

A slight generalization of a generic vector projection representation $R_w = f \circ H_w$ defined in equations 15 and 16 is obtained by using a less rigid definition of the bias parameters λ in H_w. Instead of requiring $\lambda = \langle D, w \rangle$, instead let λ be any values which separate the ordered projections of the m points of D in the sense that

[3] Viewing the interpolating function y as a function on the line with direction w and coordinate λ in the original data space $\mathbb{R}^n$, we see that it is piecewise linear. By part six of the lemma the pairwise slopes of the piecewise linear function joining successive pairs of points $(\lambda_1, y_1), ..., (\lambda_m, y_m)$. determine the first $m-1$ values of c.

$$\lambda_i \leq \langle x_i, w \rangle \leq \lambda_{i+1} \; for \; 1 \leq i \leq m-1 \tag{18}$$

Denote this matrix by $H_{w,\lambda_{D,sep}}$ and the representation function by $R_{w,\lambda_{D_{sep}}} = f \circ H_{w,\lambda}$. The matrix for $R_{w,\lambda_{D_{sep}}}$ remains upper triangular but the i^{th} diagonal entry is positive, not zero, if $\lambda_i < x_i$, so the representation of the data points x_i remain independent. This effectively determines a translation of the y_i values, slightly changing the explicit formulas in the GVP Representation Lemma 2.

3.5 A Two-Layer Neural Network Interpolating xor

Previously we proved that no one-layer neural network function interpolates the *xor* function. Recall the *xor* function is defined in the data set $D = \{(0,0),(1,0),(0,1),(1,1)\}$ and its values of *xor* are (0,1,1,0),respectively. We will explicitly construct one of the two-layer neural network functions interpolating *xor* guaranteed by the corollary. We will use the vector $w = (1,1)$ because it is geometrically the most natural choice. $\langle D, w \rangle = \{0,1,1,2\}$. The second and third points of D project onto the same point of the line determined by W, and *xor* has the same value on both of these points so the induced function $\bar{Y}_w$ is well-defined. Part 2 of the corollary applies. We will only need to use one of two points to define the neural network function. Thus we will now assume $D = \{(0,0),(1,0),(1,1)\}$, so $m = |D| = 3$ and $n = 2$. Then $< D, W >= (\lambda_1, \lambda_2, \lambda_3) = (0,1,2)$ and $xor(D) = (y_1, y_2, y_3) = (0,1,0)$. The first layer function is defined by the $m+1 = 4$ by $n+1 = 3$ matrix

$$H_w = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & -1 \\ 1 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

The affine representation of the data points is

$$D = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{20}$$

So

$$H_w D = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{21}$$

showing the $(i,j)^{th}$ entry is $\lambda_j - \lambda_i$ and

$$R_w(D) = f(H_w D) = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{22}$$

showing the representation $R_w(D)$ is an upper triangular matrix with zeros on and below the diagonal of the 3×3 matrix formed by the first three rows because the λ's are increasing.

The second layer function is defined by the 1 by $m+1 = 4$ matrix

$$Y = [c_1, c_2, c_3, c_4] = \big[[1, -2, ?, 0]\big] \tag{23}$$

showing $c_4 = y_1$ and

$$\frac{y_j - y_{j-1}}{\lambda_j - \lambda_{j-1}} = c_1 + ... + c_{j-1} \text{ for } j = 2, 3 \tag{24}$$

There are no conditions on c_3 and none are needed because the m^{th} (i.e.third) row of $R_w(D)$ is all zeros. Multiplying and evaluating gives $Y(R_w(D)) = xor(D) = [0, 1, 0]$. showing the 2 layer neural network function Y interpolates *xor*.

A second layer function $Y = (1, -2, 0, 0)]$ can also be validated geometrically. The first layer function maps the four *xor* data points in $\mathbb{R}^2$ to the three points in $\mathbb{R}^3$ shown in equation 22. The line in $\mathbb{R}^3$ through the origin with positive direction $(1, -2, 0)$ is perpendicular to the $(2, 1, 0) - (0, 0, 0)$ the vector in $\mathbb{R}^3$ joining the images of the two original points in $\mathbb{R}^2$with *xor* value 0 so their projections on the line are equal and the value of the inner products is 0.The projection of the images of *xor* points with value 1 is 1(when the line is parametrized as multiples of the the vector $(1, -2, 0)$. Hence the values of the projections equal the values of the *xor* function on the images of the *xor* points. Thus even though there was no line in $\mathbb{R}^2$ with respect to which the *xor* function was monotonic after projection, we have constructed a two-layer neural network which maps the four *xor* points (with two distinct values) to two distinct points in $\mathbb{R}^3$ whose *xor* values equal the values of the inner product with the vector determining the direction and parametrization of the line.

4 Gradient Descent for One Layer Networks

Given a non-negative real-valued function defined on a finite data set and a one-layer neural network function, we define the loss function and prove a useful formula for the gradient of the loss function. We also prove that gradient descent must eventually enter the positive affine cone determined by the data points in order for the loss function to reach zero. Let $\bar{x}_k \in \mathbb{R}^{n+1}$, $k = 1, ..., m$ denote the positive affine coordinates for the points in D The *positive affine cone* $A \subset \mathbb{R}^{n+1}$ *determined by the data set* $D \subset \mathbb{R}^n$ is defined as

$$A = \{h \in \mathbb{R}^{n+1} : \langle h, \bar{x}_k \rangle > 0 \ : \ k = 1, ..., m\} \tag{25}$$

Lemma 3. ***Gradient Formula and Loss Bound***
Given a one-layer neural network function $Y : \mathbb{R}^{n+1} \to \mathbb{R}$, $Y = f \circ h$, $h = (h_1, ..., h_{n+1})$, and a non-negative real-valued function $\bar{Y}$ defined on a finite subset $D \subset \mathbb{R}^n$ with affine coordinate representations $\bar{x}_1, ..., \bar{x}_m$, $m \leq n$

1. *The gradient ∇L*

$$\nabla L = (\frac{\partial L}{\partial h_1}, ..., \frac{\partial L}{\partial h_{n+1}}) \tag{26}$$

of the standard loss function L

$$L = \sum_{k=1}^{m} (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k))^2 \ for x_k = x_1, ..., x_m \in D \tag{27}$$

is defined if none of the inner products $< h, \bar{x}_k >$ are zero.

$$\nabla L = 2 \sum_{k:\langle h, \bar{x}_k \rangle > 0} (\langle h, \bar{x}_k \rangle - \bar{Y}(\bar{x}_k)) \cdot \bar{x}_k \tag{28}$$

If $h \notin \overline{A}$, the closure of the positive affine cone A,

$$L(h) \geq \sum_{k:<h, x_k><0} \bar{Y}(\bar{x}_k)^2 \tag{29}$$

2. *Gradient descent will converge to a function with zero loss only if it eventually moves through $h \in A$.*

Proof. The definition of Relu, equation 2, implies

$$L = \sum_{\langle h, \bar{x}_k \rangle > 0} (f(\langle h, \bar{x}_k \rangle) - \bar{Y}(\bar{x}_k))^2 + \sum_{\langle h, \bar{x}_k \rangle \leq 0} \bar{Y}(\bar{x}_k)^2 \tag{30}$$

which implies the bound on the loss function in part two of the lemma. This also implies that the loss can go to zero only if the gradient descent moves through affine transformations $h \in A$. For part one of the lemma: computation of the gradient exploits the chain and product rule.

$$\nabla L = \sum_{k=1}^{m} \nabla((Y(\bar{x}_k) - \bar{Y}(\bar{x}_k))^2) \tag{31}$$

$$\nabla (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k))^2 = 2 \cdot (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k)) \cdot \nabla (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k)) \tag{32}$$

$$\nabla (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k))^2 = 2 \cdot (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k)) \cdot \nabla (Y(\bar{x}_k) \tag{33}$$

Next

$$\nabla Y(\bar{x}_k) = f'(\langle h, \bar{x}_k \rangle) \cdot \nabla (\langle h, \bar{x}_k \rangle) \tag{34}$$

Since

$$\frac{\partial \langle h, \bar{x}_k \rangle}{\partial h_j} = e_j \cdot \bar{x}_k \tag{35}$$

where e_j is the unit vector for the j^{th} coordinate

$$\nabla Y(\bar{x}_k) = f'(\langle h, \bar{x}_k \rangle) \cdot \bar{x}_k \tag{36}$$

We are assuming that $f = Relu$ so

$$f'(x) = 0 \, for \, x < 0 \tag{37}$$
$$f'(x) = 1 \, for \, x > 0 \tag{38}$$
$$f'(0) = undefined \tag{39}$$

Substituting into equations31 and 33

$$\nabla L = 2 \sum_{k=1}^{m} (Y(\bar{x}_k) - \bar{Y}(\bar{x}_k)) f'(\langle h, \bar{x}_k \rangle) \cdot \bar{x}_k \tag{40}$$

Since $Y = f \circ h$ (6) ,

$$\nabla L = 2 \sum_{k=1}^{m} (f(\langle h, \bar{x}_k \rangle) - \bar{Y}(\bar{x}_k)) f'(\langle h, \bar{x}_k \rangle) \cdot \bar{x}_k \tag{41}$$

If none of the inner products are zero, $\langle h, \bar{x}_k \rangle \neq 0$ for all $k = 1, ..., m$, the gradient is defined and, applying equations 2 and 39, equal to

$$\nabla L = 2 \sum_{\langle h, \bar{x}_k \rangle > 0} (\langle h, \bar{x}_k \rangle - \bar{Y}(\bar{x}_k)) \cdot \bar{x}_k \tag{42}$$

□

Corollary 2. ***Gradient Descent Operator***
Let A denote the affine cone $A = \{\bar{x} : \cap_{1 \leq k \leq m} \langle \bar{x}, \bar{x}_k \rangle > 0\}$ *and let* ∂A *denote its boundary.*

1. *If there exists a non-negative vector* $c \in \mathbb{R}^{n+1}$*, such that* $\bar{Y}(\bar{x}_k) = \langle c, \bar{x}_k \rangle$*,* $k = 1, ..., m$*, then* $c \in A \cup \partial A$ *and the loss gradient formula for* $h \notin \partial A$ *is*

$$\nabla L(h) = 2 \sum_{1 \leq k \leq m : \langle h, \bar{x}_k \rangle > 0} (\langle h - c, \bar{x}_k \rangle \cdot \bar{x}_k \tag{43}$$

   The gradient of the loss is undefined on the boundary of the affine cone.
2. *Let*

$$M_h = \sum_{1 \leq k \leq m : \langle h, \bar{x}_k \rangle > 0} (\bar{x}_k \bar{x}_k^T) \tag{44}$$

   then for $h \notin \partial A$

$$\nabla L(h) = 2 M_h (h - c) \tag{45}$$

3. *M_h is symmetric. If the x_i are non-negative, M_h has non-negative entries and non-negative eigenvalues. If the x_i are computed by a neural net function, they are non-negative. If x_i $i = 1,...,m$ are independent, M_h has rank m.*
4. *Let V denote the subspace spanned by the vectors $\bar{x}_k$, $k = 1,...,m$ with orthogonal complement $V^\perp$, Assume $h \notin \partial A$. If $h \in V^\perp$, $\nabla L(h) = 0$. If $h \in V$, $\nabla L(h) \in V$.*
5. *Let $GD(h,\varepsilon,i)$, $i = 1,.....$ denote the result of iterating gradient descent for the loss function $L = \sum_{i=1}^{m}(Y(\bar{x}_i) - \bar{Y}(\bar{x}_i))^2$, beginning at $h \notin \partial A$, using positive parameter $\varepsilon < 1$, i times.*

$$GD(h,\varepsilon,1) - c = (I - 2\varepsilon M_h)(h - c) \tag{46}$$

6. *Let α_{max} and α_{min} denote the maximum and minimum eigenvalues of M_h, respectively. For $\varepsilon > 0$ in the range $1/4 < 2\varepsilon\alpha_{max} < 1/2$ and $h \notin \partial A$*

$$GD(h,\varepsilon,1) - c\| < (1 - 2\varepsilon\alpha_{min})\|h - c\| < (1 - \frac{\alpha_{min}}{4\alpha_{max}})\|h - c\| \tag{47}$$

7. *If M_h is positive definite ($\alpha_{min} > 0$) on V and $h \in A$*

$$GD(h,\varepsilon,i) - c = (I - 2\varepsilon M_h)^i(h - c) \tag{48}$$

The gradient descent converges to c.

Proof. For part one $c \in A \cup \partial A$ because $\bar{Y} \geq 0$. The formula 43 follows from the lemma by substitution. For part two, viewing the $\bar{x}_k$ as column vectors, each term can be rewritten as

$$\langle h - c, \bar{x}_k\rangle \cdot \bar{x}_k = \bar{x}_k \cdot \langle \bar{x}_k, h - c\rangle = (\bar{x}_k \bar{x}_k^T)(h - c) \tag{49}$$

so summing gives formula 45. Part three follows by from the formula for M_h, the fact that non-negative symmetric matrices have non-negative eigenvalues and the SVD decomposition for the matrix X whose columns are the $x_i's$. The first statement in part four is true because the sum in formula 43 is over the empty set. The second statement in Part four is true because for any vector $v^\perp \in V^\perp$, $\langle v^\perp, x_k\rangle = 0$, implying $\langle v^\perp, (\bar{x}_k \bar{x}_k^T)(h - c)\rangle = 0$. For part five, the definition of gradient gives

$$GD(h,\varepsilon,1) = h - \varepsilon\nabla L = h - 2\varepsilon M_h \cdot (h - c) \tag{50}$$

The vector difference between $GD(h,\varepsilon,1)$ and c is

$$GD(h,\varepsilon,1) - c = (I - 2\varepsilon M_h)(h - c) \tag{51}$$

For part six, the entries of D_h are in the interval $[1 - 2\varepsilon\alpha_{max}, 1 - 2\varepsilon\alpha_{min}]$. Choose $\varepsilon : 1/4 < 2\varepsilon\alpha_{max} < 1/2$. The right hand inequality ensures that the entries of D_h are positive. The left hand inequality implies $\frac{\alpha_{min}}{4\alpha_{max}} < 2\varepsilon\alpha_{min}$ and an upper bound on the entries of $(I - 2\varepsilon D_h)$. For part seven: M_h is symmetric, so can be diagonalized $M_h = U_h D_h U_h^T$. Hence we may assume after a change of coordinates $\bar{x} = D^{1/2}Ux_1$ that M_h is the identity matrix. Formula 46 shows that $GD(h,\varepsilon,1)$ is on the line seg-

ment joining c and h, so is still in the convex affine cone, and is closer to c. Hence the result can be iterated. Gradient descent converges to c because $(I-2\varepsilon M_h)^i$ converges to the zero matrix. Note that $M_h = U_h D^{1/2} * (U_h D^{1/2})^T$ so M_h is a kernel matrix. □

Note that part three of theorem implies that M_h for GVP representations is positive definite on the subspace spanned by the representation vectors. Hence gradient descent converges to c for GVP neural network functions providing the initial condition h is in the affine cone determined by the representation points.

5 Generalizability

5.1 Empirical Generalizability

Given a non-negative function $\bar{Y}$ on a finite data set $D \subset \mathbb{R}^n$, a partition of D into a training set and a test set $D = D_{tr} \cup D_{te}$, a non-negative function $Y : \mathbb{R}^n \to \mathbb{R}$ learned using only the training data (and not the test data), and a real-valued loss function L defined on the parameters of the model family and $\mathbb{R}^n$, Y empirically generalizes well if the expected sample error on the test data is "small". This error is likely to be larger than the expected error on the training data because the model was learned using the training data.

Defining the expected sample errors on the training data and test data, respectively, by

$$E_{te}(Loss) = |D_{te}|^{-1} \sum_{x \in Test} Loss(Y(x), \bar{Y}(x)) \tag{52}$$

$$E_{tr}(Loss) = |D_{tr}|^{-1} \sum_{x \in Train} Loss(Y(x), \bar{Y}(x)) \tag{53}$$

the *empirical generalization error* GE_{emp} is formally defined by the equation

$$GE_{emp} = E_{te}(Loss) - E_{tr}(Loss) \tag{54}$$

Overfitting is said to occur when GE_{emp} is "too large" [15](Section 5.2).

5.2 Minimizing GVP generalization error

We will analyze the empirical generalization error for the family of two-layer GVP neural networks functions $Y_w :\subset \mathbb{R}^{n+1} \to \mathbb{R}$, where w varies over the set of vectors $w \in \mathbb{R}^n$ which are generic for D. $Y_w = Y_{tw}$ for $t > 0$ by the Neural Network Interpolation Corollary 1 , it suffices to analyze the empirical generalization error of Y_u for u in the unit sphere $S^{n-1} \subset \mathbb{R}^n$, u generic for D. For each function Y_u in this family the

empirical error equals the loss on the test set, $GE_{emp}(Y_u) = E_{te}(Loss(Y_u))$, because GVP Neural Network functions interpolate $\bar{Y}$ on the training set D_{tr}. For example, if D_{te} contains only one point z, $GE_{emp}(Y_u) = (Y_u(z) - \bar{Y}(z))^2$. Our goal is to prove that the empirical error is minimized for some Y_u, $u \in S^{n-1}$ and to prove that the minimization problem is convex. In directions u for which the empirical error of Y_u is minimized the projections of the test set are "better aligned" with the projections of the training set.

Proving this is slightly more complicated than saying a continuous function on a compact set achieves a minimum because the function

$$F(u) = E_{te}(Loss(Y_u)) \tag{55}$$

is not defined everywhere on S^{n-1} . It is undefined when Y_u is undefined, i.e. at the set *Disc* of points $u \in S^{n-1}$ such that two distinct training points x_i and x_j with different values of $\bar{Y}$ are projected onto the same point on lines in the direction of u. The set of discontinuities *Disc* is contained in the larger set $G_D \cap S^{n-1}$ of generic points for D on the unit sphere. Thus $F(u)$ is defined on $S^{n-1} - Disc \supset S^{n-1} - G_D$ The latter set, in general, consists of multiple connected components C_α, The union of the closure of the components is S^{n-1}, $\cup\overline{C_\alpha} = S^{n-1}$

Lemma 4 (Mininimizing GVP Generalization Error). *Given a non-negative function $\bar{Y}$ on a finite data set $D \subset \mathbb{R}^n$ with a partition of D into a training set and a test set $D = D_{Tr} \cup D_{Te}$, and two layer GVP neural network functions Y_u, $u \in S^{n-1} - Disc$: The function $F(u) = E_{te}(Loss(Y_u))$*

1. *computes empirical generalization error*
2. *is a continuous function on $S^{n-1} - Disc$ and hence a continuous function on each of the connected components C_α of $S^{n-1} - G_D$.*
3. *determines a continuous function with finite values on the closure of each of the connected components $\overline{C_\alpha}$.*
4. *determines a possibly multi-valued function on G_D (with finite values) and a single-valued function on $S^{n-1} - G_D$*
5. *attains a finite minimum on S^{n-1}.*
6. *is a geodesically convex function on each geodesically convex set $\overline{C_\alpha}$.*

Proof. Consider the trivial fiber bundle $S^{n-1} \times \mathbb{R}$. For any point $x \in \mathbb{R}^n$ the function $u \to \langle x, u \rangle$ is continuous and hence is a continuous global section of the fiber bundle. In particular, $\langle x_j, u \rangle$ is a continuous global section for each point x_j in the test set D_{te} so $F(u) = E_{te}(Loss(Y_u))$ is continuous on $S^{n-1} - Disc$. The sections intersect only at points in $G_D \cap S^{n-1}$, so in each non-empty connected component of $S^{n-1} - G_D$, the set of global sections $\{\langle x_j, u \rangle, x_j \in D_{te}\}$ do not intersect and are in a particular order. In other words each non-empty connected component of $S^{n-1} - G_D$ determines a total order on the set of training points. In each component, the point evaluation functions $u \to Y_u(x)$ are continuous in u and hence continuous sections on each component C_α of $S^{n-1} - G_D$. Hence these continuous sections $u \to Y_u(x)$, $u \in C_\alpha$ are bounded on each component and determine continuous functions $u \to Y_{u,\overline{C_\alpha}}(x)$ on the closure of each component. Note that this implies the evaluation functions

$u \to Y_u(x)$ extend to multi-valued functions (with finite values) on *Disc*. In particular, for each test point x_j, the evaluation functions $u \to Y_u(x_j)$ extend to multi-valued functions (with finite values) on *Disc*. Hence the evaluation function for the empirical expected loss function 52

$$u- > |D_{te}|^{-1} \sum_{x_j \in D_{te}} Y_u(x_j) \tag{56}$$

extends to a multi-valued function (with finite values) on G_D and is continuous on the closure of each component C_α. Hence it achieves a minimum on S^{n-1} and its argmin set may include points of G_D.

The closure of each of the connected components $\overline{C_\alpha}$ is a geodescially convex subset of the sphereS^{n-1}. On each component, the evaluation function for the empirical expected loss functions is convex, so the argmin set can be found by performing gradient descent or even by solving constrained constrained normal equations on each component and then combining the results. □

For directions u in the argmin set of F the projections of the test set onto u, $\langle D_{te}, u\rangle$, are more well-aligned with the projections of the training set onto u, $\langle D_{tr}, u\rangle$, in the sense that they are closer to training points with similar values. For this family of neural network functions these projections come most accurately linearly interpolating nearest neighbor predictions.

5.3 Final Model and Parallel Architecture

The final model could be a model with the minimum generalization error over all the training and test set pairs analyzed. Another approach is to notice that training sets of size $n-k$ obtained using a "leave out k" strategy all have the same size; a finite set of such training samples are all interpolated on the $n-k-1$ dimensional manifold of directions in S^{n-1-k} which are the complement of a set of hyperplanes; the proof of lemma 4 applies to the sum of their test errors (i.e. the sum of their generalization error is minimized) so there is a direction u_{min} minimizing the sum of their generalization errors. The average of such a set of models thus interpolates the test set and has a minimum generalization error. Averaging over "leave out k"' , even when $k = 1$ is conceptually justified by the stability theorem based on a leave out one criterion which generalizes generalizability [26]. We also note that leave out 1 generalizabilty is the priniciple underlying differential privacy [12]. In general we cannot use the same approach to minimize average generalization for an average of a set of models, where test sets have varying sizes they will have zero interpolation error only on the intersection of the test sets which may be empty.

5.4 Theoretical Generalizability

The theoretical definition of generalization error requires knowledge of the joint distribution for $\mu(Y,\bar{Y})$ on $\mathbb{R}^{n+1}$. The expected generalization error is defined as the expected value of the Loss function.

$$GE = E_{\mu}(Loss(Y,\bar{Y})) \tag{57}$$

In the case of the standard loss function :

$$GE = E_{\mu}(Y(x) - \bar{Y}(x))^2 \tag{58}$$

The theory assumes that the training set is generated by *iid* sampling from the joint distribution. The model is learned from this training set (so the expected error should be smaller on the training set). Theoretically the model generalizes if as the sample size increases to infinity, the expected empirical generalization error converges to the generalization error defined by equation 57 [26]. In [26] a new type of general stability condition ('leave out one stability') was defined and proved to be sufficient for generalization. The result did not require an explicit connection with the properties of the joint distribution.

In practice, $\bar{Y}$ is not known on the universe of the data set, the joint distribution μ is unknown, the sample is not generated *iid* and sample numbers and sizes are restricted because of unavailability of data, time, and computation effort.

5.5 Data-Determined Generalizability Measures

We propose that multiscale representations of approximations to measures could be useful for better understanding generalizability properties can be computed from the data set, test and training samples, and training and test errors for sets of models. (A measure is just a positive multiple of a probability distribution). The particular multi-scale non-parametric representation we propose is the *product formula representation* [4] [13] [28]. Because the parameters of the representations are unique and can be computed using simple formulas which do not involve optimization, the measures for data sets, samples, and models can be compared, visualized and, if desired, averaged. Noisy variants of the representations can also be computed and analyzed. The representation parameters will help pinpoint the localities of the differences. As the models are used on new data sets after a testing and training process, analogous measures could be computed and compared to understand if the real world application data is significantly different than the original testing and training data.

5.5.1 Product Formula Representation

The product formula representation (PFR) is defined for any positive measure defined on sets X with an ordered dyadic binary tree structure. Left and right subsets $L(S)$ and $R(S)$ of a set S are recursively defined. The recursion may be infinite. The measures may therefore be represented as an ordered binary tree (possibly of infinite depth), whose nodes correspond to left or right subsets of their parents. The root node corresponds to the universe set X. Each measure μ on the sigma algebra generated the these sets is uniquely described by the total measure $\mu(X)$ and a parameter at each non-leaf node S, called the *product coefficient parameter* a_S defined by

$$a_S = \frac{\mu(L(S)) - \mu(R(S))}{\mu(S)} \; if \; \mu(S) \neq 0 \tag{59}$$

and the convention

$$a_S = 0 \; if \; \mu(S) = 0 \tag{60}$$

By construction, $-1 \leq a_S \leq 1$; it's value is the difference between the conditional distributions of the measures of the left and right subsets given the measures of the parent. It this difference is large, i.e. close to 1 or -1 it indicates a locality where the concentration of the measure change a lot. For each dyadic set S, represented as a node at distance n from the root node of the tree X, the unique path to S from the X through nodes S_i $i = 1...n$ can be described as a sequence of ± 1 values dir_{S_i} , with -1 indicating that S_i is left child and 1 indicating that S_i is a right child. The product formula for the measure of the set S is

$$\mu(S) = \frac{\mu(X)}{2^n} \prod_{i=1}^{n} (1 + dir_{S_i} a_{S_i}) \tag{61}$$

The theory applies if the the measures of the dyadic sets are non-negative and the total measure of the set X is positive. This formula converges in the infinite case [4] (Lemma 2.1). This can be viewed as a normalized formalization of the chain rule for probability measures on sets X with a binary tree structure. The paper [4] (Section 2.5) describes a number of examples, including applications of this representation to real world data and visualization of the resulting parameters. In applications to real world data, the measure parameters are computed only for a tree of a fairly small depth. The paper [4] proves a product formula convergence lemma, a visualization theorem, and a noise model theorem.

A natural application is representation of measures on the unit interval with the usual binary tree structure on dyadic intervals. Another natural application is measures on sets X with an ordered collection of subsets S_i (e.g. real numbers with n digits whose i^{th} binary digit is 0). Then a set S at distance n from the root of the tree is the intersection of sets S_i or the complement $X - S_i$ depending on whether the path direction is left or right. Here we can exploit both types of applications.

5.5.2 Proposals for GVP Understanding using PFR's

The GVP generalization error for a family Y_u is a function on S^{n-1}. It is a positive function since it equals the sample average of the loss on the test set. Using spherical coordinates, the sphere can be represented as a product of $n-2$ finite intervals $[0,\pi]$ and a finite interval $[0,2\pi]$. A binary tree could be constructed by ordering the coordinates and successively dividing the different coordinate directions in half repeatedly. The GVP generalization error could be computed for a sample of directions $u \in S^{n-1}$ and the average of the values could be computed for each of the dyadic sets at the finest scale. The product coefficient parameters can be computed bottom up from the histograms determined by the averages at the finest scale. Experiments with the method could be conducted using small values of n. For larger values of n, only a few scales would need to be initially computed. The results product coefficients and visualizations could be compared for different families of Y_u determined by the same data set, using different training and test sets. They could also be compared for different data sets.

The model Y_u and $\bar{Y}_u$ could be compared on the training set D_{tr} by restricting them to a finite interval I on the line containing the points at which $\bar{Y}_u$ is defined. There are two alternative methods for defining a binary tree structure on the interval. First, a dyadic decomposition of this interval into (2^{-s}) equal subintervals for $s = 1,...,k$ determines a binary tree structure. A measure for Y_u is defined by integrating Y_u on each subinterval and viewing the measure of the whole interval as one. A measure approximating $\bar{Y}_u$ can be defined by averaging the values of $\bar{Y}_u$ in each subinterval. The product coefficients for each can be compared numerically and visually. Since the values of the product coefficients are constrained to be in the interval $[-1,1]$, many visualization methods can be used. A simple one is a daywheel as shown in REF. These two measures can be compared with the measure for $\bar{Y}_u$ determined by the test set D_{te} or the whole set D. The measures for multiple directions u could be compared as could the measures for multiple divisions of D into training and test subsets. The measures could also be averaged. Counting measures can also be computed for the training set, the test set and the entire data set. The counting measure for new data sets to which the model is applied could be computed on a going forward basis and compared with the measures used to construct and test the model.

An alternative method for defining a binary tree structure on the interval $I = [\lambda_0, \lambda_{m+1}] \subset tu$ is to use the separation parameters defined in Lemma 2 to define disjoint $m+1$ ordered subsets of I by

$$S_1 = \{tw : t < \lambda 1\} \tag{62}$$

$$for\ i = 2,..,m : :\ S_i = \{tw : \lambda_{i-1} \leq t < \lambda_i\} \tag{63}$$

$$S_{m+1} = \{tw : \lambda_m \leq t\} \tag{64}$$

a set S at distance $1 \leq n \leq m+1$ from the root of the binary tree is the intersection of sets S_i or the complement $X - S_i$, $i = 1,...,n$ depending on whether the path direction is left or right.

Lemma 5. *For this binary tree structure, the product coefficient for the i^{th} set on rightmost path, $i = 1,...,m$, is*

$$a_i^R = \frac{\mu(S_{i+1}) - \sum_{k=i+2,..,m+1} \mu(S_k)}{\sum_{k=i+1,...,m+1} \mu(S_k)} \tag{65}$$

All of the other product coefficients are zero or -1. If the measures of all of the sets S_i are positive, the product coefficients are independent of the measure.

Proof. The sets corresponding to the nodes on the rightmost path R in the binary tree are: $I - S_1$, $(I - S_1) \cap (I - S_2)$, ..., $\cap_{i=1...,m+1}(I - S_i)$. Thus the i^{th} set on the right most path is $\cup_{j=i+1,...,m+1} S_j$. The last set corresponding to the last node on the rightmost path is therefore empty. The product coefficient for the i^{th} set on this path, $i = 1,...,m$, is

$$a_i^R = \frac{\mu(S_{i+1}) - \sum_{k=i+2,..,m+1} \mu(S_k)}{\sum_{k=i+1,...,m+1} \mu(S_k)} \tag{66}$$

Product coefficients are not computed for leaf sets (at level $m+1$). Since the sets are ordered and disjoint, the other node sets are either empty or equal to S_i. If a node set equals S_i, all it descendants on the right most path emanating from the node are equal to S_i and all other descendants are empty. Thus the other product coefficients are either 0 or -1. If $\mu(S_i) > 0$ product coefficients for all of its descendants on the right most path emanating from a node set equaling S_i are -1 regardless of the measure; otherwise they are 0.

6 Reprise of the GVP Model and its Properties

The reprise of the model is as follows: The first layer of a GVP neural net function is determined by the data set and consists of set of projections onto a line generic for the training data set translated by the bias determined by the projections of the training data, followed by composition with the Relu function which zeros out negative values. This first layer computes a linearly independent representation of the training data, which due to the Relu function, is easily an invertible upper triangular matrix enabling to explicit computation of the weights and bias defining the second layer. Because the representation computed by the first layer for the training set consist of independent vectors, it determines a data dependent positive definite kernel. We prove that gradient descent for the standard loss function also converges to an exact solution when the gradient descent is initialized to be in the positive affine cone determined by by representation of the data set. If the initialization is the orthogonal complement of the representation, no descent will occur because the gradient at this intialization is 0. GVP neural nets exactly interpolate the training data (i.e. have zero training error); they are homogeneous of degree 0. Hence GVP neural nets are parameterized by the $n-1$ dimensional submanifold of unit vectors on the sphere in the n dimensional training data space which are in the complement of a finite set of

hyperplanes where two training data points project to the same point. In each each component of the model parameter sub-manifold, the projections of the training data are ordered in a particular way. In other words, we can completely characterize the manifold of parameters for optimal models with zero training error. We prove that the test error function extends to a possibly multi-valued function on the sphere S^{n-1} that achieves a global minimum, and is a geodesically convex function. Hence the variation of the empirical generalization error on the optimal parameter sub-manifold can be understood and parameters minimizing the generalization error can be computed. Geometrically, the minimum of the empirical generalization error for this particular training and test data set occurs for generic directions for which the projection of the training set is optimal with respect to the projection of the test set for a nearest neighbor interpolation prediction. The empirical generalization error can be minimized for the average model of a set of "leave out k" samples. Multi-scale non parametric representations of the generalization error measures resulting from this statistical learning process can be computed to summarize them. The same technique can be used to compute representations of the training and test sets and used later to compare real world data sets with the original training and test data.

7 Summary and Research Directions

We defined a simple data dependent geometrically interpretable two-layer architecture for real-valued functions which result in a large set of models which exactly interpolate a given training set and proved that the Relu function ensures that parameters for the second layer can easily be computed explicitly. The set of interpolation models for a single sample is parameterized by a submanifold of the sphere in the data space which is the complement of a finite set of hyperplanes determined by the data. The empirical generalization error function is geodesically convex on the sphere in the data space so in principle the parameters of the models minimizing generalization error can be computed. Analysis revealed that the models exploits principles that recent theoretical research has been shown to be effective: adaptation of the architecture to the data, positive homogeneity, parallel architectures, unbounded width, zero training error, and data dependent kernels. Powerful deep learning techniques including stochastic gradient descent, regularization and convolution are not utilized in this simple model. The hope is that this paper will motivate definition of other geometric models accompanied by proofs, which will both simply demonstrate some of the principles powering the new advances in deep learning research using relatively small data sets and the subtleties and strengths of more complex deep learning models and algorithms .

While the original goal was increased understanding, it also seems the model merits more research and may apply more generally. Experiments should be conducted using the model and the results should be visualized. The methods for empirical generalization analysis need to be detailed and connections with total variation minimization and nearest neighbor interpolation explored. t may be possible that

generalization analysis can exploit the random linear projection algorithm for nearest neighbors [22].Gradient descent for the more general two layer model, which doesn't restrict the weights to be the specific functions of the data used in this paper, should be analyzed to see if the weights converge to the functions of data used in the GVP model. Fortunately, it seems that experimental research and applications using the GVP model can be conducted using small data sets and straightforward algorithms.

Acknowledgements The author thanks the National Science Foundation for support under grant number 1934924 to Rutgers University and ICERM for hosting and supporting the July, 2019 WiSDM Research Collaboration Workshop in Mathematics and Data Science.

References

1. Arora, S., Du, S., Hu, W., Li, Z., Wang, R.: Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. In ICML (2019).
2. Arora, S., Du, S., Hu, W., Salakhutdinov, R., Wang, R.: On exact computation with an infinitely wide neural net. In NeurIPS (2019).
3. Bach, F.: Breaking the Curse of Dimensionality with Convex Neural Networks. J Mach Learn Res 18, 1-53 (2017).
4. Bassu, D., Jones, P.W., Ness, L., Shallcross, D.: Product Formalisms for Measures on Spaces with Binary Tree Structures: Representation, Visualization and Multiscale Noise. submitted. https://arxiv.org/abs/1601.02946.
5. Belongie, S., Fowlkes, D. Chung, F. Malik, J.: Spectral Partitioning with Indefinite Kernels Using the Nystrom Extension. IEEE T Pattern Anal, Vol. 26, No. 2 (2004).
6. Bengio, Y., Le Roux, N., Vincent, P., Delalleau, O., Marcotte, P. : Convex neural networks. In NIPS (2005).
7. Bertozzi, A.: Graphical Models in Machine Learning, Networks, and Uncertainty Quantification. Proc. Int. Cong. of Math 2018, Vol. 3, 3853-3880 (2018).
8. Chizat, L., Bach, F. : On the Global Convergence of Gradient Descent for Over-parameterized Models using Optimal Transport. In NIPs (2018).
9. Cooper, Y. : The loss landscape of overparametrized neural networks. https://arxiv.org/abs/1804.10200
10. Cybenko, G. : Approximation by superpositions of a sigmoidal function. Math Control Signal, 2, 303-314 (1989).
11. Du, S., Zhai, X., Poczos, B., Singh, A.: Gradient descent provably optimizes over-parameterized neuralnetworks. arXiv preprint arXiv:1810.02054 (2018).
12. Dwork,C., McSherry, F., Nissim,K., Smith, A. : Calibrating Noise to Sensitivity in Private Data Analysis. In TCC, Springer (2006). The full version appears in J. of Privacy and Confidentiality, 7 (3), 17-51.
13. Fefferman, R., Kenig, D., Pipher, J.: The Theory of Weights and the Dirichlet Problem for EllipticalEquations. Ann Math. no. 134, pp. 65-124 (1991).
14. Guilhoto, L.F.: An Overview Of Artificial Neural Networks for Mathematicians. http://math.uchicago.edu/ may/REU2018/REUPapers/Guilhoto.pdf.
15. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. The MIT press (2016).
16. Haeffele, B.D., Vidal, R.: Global Optimality in Neural Network Training. CVPR (2017).
17. Hornik, K., Stinchcombe, M., White, H: Multilayer feedforward networks are universal approximators. Neural Networks, 2, 359-366 (1989).
18. Jacot, A., Gabriel, F., Hongler, C. : Neural tangent kernel: Convergence and generalization in neural networks. arXiv preprint arXiv:1806.07572, 2018.

19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In NIPS(2012).
20. Kunin, D., Bloom, J., Goeva, A., Seed, C.: Loss Landscapes of Regularized Linear Autoencoders. PMLR 97:3560-3569 (2019). http://proceedings.mlr.press/v97/kunin19a.html
21. Leshno, M., Lin, V.Y., Pinkus, A., Schocken, S.: Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks, 6, 861-867 (1993).
22. Li ,K. , Malilk, J.: Fast k-nearest neighbour search via Dynamic Continuous Indexing. In ICML, pp. 671-679 (2016).
23. Li, K., Malik, J.: Fast k-Nearest Neighbour Search via Prioritized DCI. In ICML (2017). https://arxiv.org/pdf/1703.00440.pdf.
24. Clark, K., Khandelwal, U., Levy, O., Manning, C.D.: What Does BERT Look At? An Analysis of BERT's Attention. In ACL (2019).
25. Mei,S., Montanari, A., Montanari, S., Nguyen,A., Nguyen, P. M.: A mean field view of the landscape of two-layers neural networks. arXiv preprint arXiv:1804.06561(2018).
26. Mukherjee, S., Niyogi, P., Poggio, T., Rifkin, R.: Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. Adv Comput Math 25: 161?193 (2006).
27. Neal, R.M.: Priors for infinite networks. In Bayesian Learning for Neural Networks. 29-53. Springer (1996).
28. Ness, L.: Inference of a Dyadic Measure and its Simplicial Geometry from Binary Feature Data and Application to Data Quality. In Research in Data Science, editors Carlotta Domeniconi and Ellen Gasparovic, Springer AWM Series (2019).
29. Olver, P. , Shakiban, C.: Applied Linear Algebra Second Edition. Springer (2018).
30. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker , L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Fan, H., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D.: Mastering the game of Go without human knowledge. Nature. 550 (7676): 354-359, 9 October (2017).
31. Shi, J., Malik, J.: Normalized cuts and image segmentation. PAMI, 22(8):888-905 (997).
32. Vidal, R.: Mathematics of Deep Learning. http://cis.jhu.edu/ rvidal/talks/learning/Tutorial-Math-Deep-Learning-2018.pdf.
33. Zhang, D., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In ICLR (2017).