A GEOMETRIC FRAMEWORK FOR ADVERSARIAL VULNERABILITY IN MACHINE LEARNING

by

Brian Bell

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF MATHEMATICS MATH.ARIZONA.EDU

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY OF APPLIED MATHEMATICS

In the Graduate College

THE UNIVERSITY OF ARIZONA

2023

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Brian Bell entitled "A Geometric Framework for Adversarial Vulnerability in Machine Learning" and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy of Applied Mathematics.

_____ Date: Type defense date here
David Glickenstein
(Chair)

_____ Date: Type defense date here
Kevin Lin
(Member)

_____ Date: Type defense date here
Marek Rychlik
(Member)

_____ Date: Type defense date here
Hoshin Gupta

(Member)

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_____ Date: Type defense date here

Dissertation Director: David Glickenstein

ACKNOWLEDGEMENTS

This document allows you to type your acknowledgements to people, groups and organizations that have helped you along the way...

For my Supportive Friends and Confidants...

Dedicated to my humerous and wonderful late mother Carrie Bell....

"*Young man, in mathematics you don't understand things. You just get used to them.*"

John von Neumann

Contents

List of Figures

List of Tables

List of Abbreviations

LAH	**L**ist **A**bbreviations **H**ere
WSF	**W**hat (it) **S**tands **F**or
ANN	**A**rtificial **N**eural **N**etwork
NTK	**N**eural **T**angent **K**ernel
ML	**M**achine **L**earning

List of Symbols

a	distance	m
P	power	W ($\mathrm{J\,s^{-1}}$)
ω	angular frequency	rad

ABSTRACT

This work starts with the intention of using mathematics to understand the intriguing vulnerability observed by Szegedy et al. (2014) within artificial neural networks. Along the way, we will develop some novel tools with applications far outside of just the adversarial domain. We will do this while developing a rigorous mathematical framework to examine this problem. Our goal is to build out theory which can support increasingly sophisticated conjecture about adversarial attacks with a particular focus on the so called "Dimpled Manifold Hypothesis" by Shamir, Melamed, and BenShmuel (2021). Chapter one will cover the history and architecture of neural network architectures. Chapter two is focused on the background of adversarial vulnerability. Starting from the seminal paper by Szegedy et al. (2014) we will develop the theory of adversarial perturbation and attack.

Chapter three will build a theory of persistence that is related to Ricci Curvature, which can be used to measure properties of decision boundaries. We will use this foundation to make a conjecture relating adversarial attacks. Chapters four and five represent a sudden and wonderful digression that examines an intriguing related body of theory for spatial analysis of neural networks as approximations of kernel machines and becomes a novel theory for representing neural networks with bilinear maps. These heavily mathematical chapters will set up a framework and begin exploring applications of what may become a very important theoretical foundation for analyzing neural network learning with spatial and geometric information. We will conclude by setting up our new methods to address the conjecture from chapter 3 in continuing research.

Chapter 1 Introduction

The primary aim of this study is to comprehend the perplexing vulnerability identified by Szegedy et al. (2014) within artificial neural networks. These models consistently admit inputs that may be geometrically close and often indistinguishable, according to users, from natural data, yet they lead to substantial changes in output. In recent years, the utilization of such models has proliferated across scientific, industrial, and personal applications. Despite their widespread and growing use, the impact of adversarial vulnerability on security, reliability, and safety in machine learning remains poorly understood. To systematically investigate this phenomenon, a robust theory and a rigorous mathematical framework are essential. We intend to construct this framework from the ground up, leveraging high-dimensional geometry tools related to curvature, implicit representations drawn from functional analysis and the theory of Hilbert spaces, optimization, and rigorous experimentation. Our overarching goal is to develop a theoretical foundation that can support increasingly sophisticated conjectures about adversarial attacks, with a particular focus on the "Dimpled Manifold Hypothesis" proposed by Shamir, Melamed, and BenShmuel (2021).

Chapter One will delve into the history and architecture of neural networks. This section will provide a brief overview of the surprising theoretical and experimental results that underpin modern machine learning. Alongside these theoretical foundations, a concise yet rigorous introduction to the practical components of a neural network model, including its training objective and considerations, will be presented. Chapter Two will concentrate on the background of adversarial vulnerability. Commencing

with the seminal paper by Szegedy et al. (2014), we will develop the theory of adversarial perturbation and attack. This chapter will explore both the theoretical and practical objectives and optimization problems related to creating adversarial examples. However, we will emphasize the establishment of careful definitions for what constitutes an "adversarial" scenario in the context of modern machine learning.

Chapter Three follows the outline of a paper which is currently under submission on the topic of measuring geometric properties in order to understand adversarial examples, and to identify certain classes of attack when they are performed. This chapter will build a theory of persistence which is related to Ricci Curvature, which can be used to measure properties of decision boundaries. These properties include the theory of persistence and how it changes while interpolating across decision boundaries, measurement of normal vectors for decision boundaries, and comparison of various interpolation trajectories between various combinations of natural classed images, adversarial images, etc... The conclusion of this chapter is a conjecture relating adversarial attacks with properties of the decision boundary defined by an arbitrary model.

Chapter Four starts from a recent theory of the neural tangent kernel. By a re-ordering of a gradient formulation for the steps taken during optimization of a model for a given loss function, a representation can be obtained as the inner product of the model gradient for a test point with the sum over the model gradients for a query point. This inner product formulation is colloquially termed a “kernel method” or a “kernel machine” which has some clearer mathematical properties than artificial neural networks do. Well-known work has demonstrated that this representation is exact for infinite-width neural networks, but has significant shortcomings for practical

finite models. This chapter outlines a recently accepted paper (Bell et al., 2023) in which this kernel method is extended by careful integration along the training steps of a model's optimization in order to produce an exact representation for finite networks which can be computed in practice, and whose numerical error can be easily controlled.

Chapter Five combines these theories and a better tool for visualizing (soft ricci curvature) near decision boundaries in order to refine the conjecture stated in Chapter Three. The main purpose of this chapter is to consolidate this conjecture as a foundation for a large body of continuing work. Based on the foundation presented within the first parts of this work, we will paint an actionable geometric picture for how adversarial examples come about and the properties of both the data and models used within modern machine-learning applications. This work will be collected into a forthcoming submission to ICLR (International Conference on Learning Representations).

1.1 Background

Artificial Neural Networks and other optimization-based general function approximation models are the core of modern machine learning (Prakash et al., 2018). These models have dominated competitions in image processing, optical character recognition, object detection, video classification, natural language processing, and many other fields (Schmidhuber, 2015). All such modern models are trained via gradient-based optimization, e.g. Stochastic Gradient Descent (SGD) with gradients computed via back propagation using theory brought to the mainstream by Goodfellow et al. (2013). Although the performance of these models is practically miraculous within the training

and testing context for which they are designed, they have a few intriguing properties. It was discovered in 2013 by Szegedy et al. (2014) that images can be generated which apparently trick such models in a classification context in difficult-to-control ways (Khoury and Hadfield-Menell, 2018). The intent of this research is to investigate these *adversarial examples* in a mathematical context and use them to study pertinent properties of the learning models from which they arise.

1.1.1 Artificial Neural Networks (ANNs)

The history of Neural Networks (NNs) begins very gradually in the field of Theoretical Neuropsychology with a much-cited paper by McCulloch and Pitts in which the mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943). This initial framework for computational cognition did not include a notion for learning, however this would be brought in the following decade with in the form of optimization and many simple NNs (linear regression models applied to computational cognition). The perceptron, the most granular element of a neural network, was proposed in another much-cited paper Rosenblatt (1958). A full 7 years would pass before these building blocks would be assembled into multilevel (deep) networks which were proposed by 1965 in a paper by Ivakhnenko and Lapa (1965). Despite much theoretical work up to this point, computing resources of the time were not nearly capable of implementing all but the simplest toy versions of these models.

By the 1960s, these neural network models became disassociated from the cutting-edge of cognitive science, and interest had shifted to their application in modeling and industrial computation. The hardware limitations of the time served as a significant barrier to wider application and the concept of the "neural network" was generally

regarded as a cute solution looking for a problem. Compounding these limitations was a significant roadblock published by Minsky and Papert (1969): A proof that basic perceptrons could not encode exclusive-or. As a result, interest in developing neural network theory waned. The next necessary step in the development of modern neural network models was an advance that would allow them to be trained efficiently with computing power available. Learning methods required a gradient, and the technique necessary for computing gradients of large-scale multi-parameter models was finally proposed in a 1970 in a Finnish masters thesis by Linnainmaa (1970). Techniques from control theory were applied to develop a means of propagating error backward through models which could be described as directed graphs. The idea was applied to neural networks by a Harvard student Paul Werbos (Werbos, 1974) and refined in later publications.

The final essential puzzle piece for neural network models was to take advantage of their layered structure, which would allow backpropagation computations at a given layer to be done in parallel. This key insight, indeed the core of much of modern computing, was a description of parallel and distributed processing in the context of cognition by McClelland, Rumelhart, Group, et al. (1986) with an astonishing 22,453 citations (a number that grows nearly every day). With these pieces in place, the world was ready for someone to finally apply neural network models to a relevant problem. In 1989, Yann LeCun and a group at Bell Labs managed to do just that. Motivated by a poorly solved practical problem – recognition of handwriting on bank checks, LeCun et al. (1989) refined backpropagation into the form used today, invented Convolutional Neural Networks 1.1.2 (LeCun, Bengio, et al., 1995), and by 1998, he had worked with his team to implement what rapidly became the industry standard for banks to

recognize hand-written numbers (LeCun et al., 1998).

It is worth noting a couple of key ingredients that led to LeCun's success. First, the problem context itself was both simple and extremely rich in training data. Second, bank checks are protected both by customers' desire to write correct checks, and their consistent formatting. Finally, the computing resources needed had only just reached sufficient specifications and indeed, LeCun's team had rare access to this level of resources. This combination along with the rapid drive toward automation at the turn of the millennium left fertile ground for this discovery. This also removed one crucial threat to this system, adversarial attacks. Banks are both protected legally from the crime of fraudulent checks, and practically by the paucity of advanced computing resources at the time. While this protected nascent machine-learning implementations, time will gradually bring technological advancements that will unseat these controlled conditions.

Through the early 2000s, along with the internet hitting its stride, neural networks have quietly become ubiquitous while remaining relegated to image recognition problems. Research and time was spent on increasing speed and efficiency. To keep these increasingly complex and structured models able to scale with the growing data available, Hinton and Bengio distinguished themselves in these middle steps. (Bengio et al., 2007; Hinton and Salakhutdinov, 2006; Hinton, Osindero, and Teh, 2006) While many observers still treated them as toy models inferior to traditional modeling, the industrial success of these models was beginning to fuel a new wave of serious theoretical and practical work in the field. A generation of household names including Collobert, Hinton, Bengio, and Schmidhuber came to distinguish themselves alongside LeCun (Coates, Ng, and Lee, 2011; Vincent et al., 2010;

Boureau et al., 2010; Hinton, 2010; Glorot and Bengio, 2010; Erhan et al., 2010; Bengio et al., 2009). With this expansion in theory came a boom in the ability to practically implement more structured and capable neural networks: recurrent networks (Mikolov et al., 2010), convolutional neural networks (Lee et al., 2009), natural language processing (Collobert et al., 2011), and Long-Short-Term-Memory (LSTM) networks originally developed by Horchreiter (a student of Shmidhuber) in 1997 (Hochreiter and Schmidhuber, 1997) to actively understand time-series, including a solution to the problem of vanishing gradients problem whereby gradients computed by back propagation which constitutes a large product of small numbers especially for early layers in deep networks. This problem naturally arises for recurrent networks and many approaches that address time-series. The solution Horchreiter provided, the addition of residual connections to past parameter states. This insight has made the LSTM one of the most cited neural networks. This in combination with the surprising result that Rectified Linear Units (ReLUs) could also solve the vanishing gradient problem allowed for much deeper and more sophisticated neural networks to be implemented than ever before.

In step with this growing theoretical interest came expansion of well-maintained libraries for working with neural networks including the very early creation of the now famous Torch (Collobert, Bengio, and Mariéthoz, 2002) and its python interface that still dominates the market-share of machine learning: PyTorch (Paszke et al., 2019) with which most of the results in this work have been computed.

While neural networks had still not breached the mainstream of pop culture, they had carved out an undeniable niche by 2009. The field of computer science was ready to test what they could do. With all of the ingredients in place, 2009-2010

saw competitions across Machine Learning (ML) tasks go viral. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) `http:/image-net.org/challenges/LSVRC/`), Segmentation of Neuronal Structures in Electron Microscopy Stacks, Traffic sign recognition (IJCNN2012) and more. The era became defined by these competitions which served to not only gain visibility for the field and its strongest participants, but also rapidly push multiple ML applications up to the point of practical utility. Schimdhuber's group (Schmidhuber, 2015), and a similar group at Google dominated many of these competitions. The cutting-edge became represented by networks like Inception v4 designed by Google for image classification which contains approximately 43 million parameters (Szegedy et al., 2014). Early versions of this network took 1-2 million dollars worth of compute-time to train. Artificial Neural Networks (ANNs) now appear in nearly every industry from devices which use ANNs to intelligently adapt their performance, to the sciences which rely on ANNs to eliminate tedious sorting and identification of data that previously had to be relegated to humans. Recently natural language processing has received its own renaissance, led by Chat-Bots based on the popular transformer architecture proposed by Vaswani et al. (2017). Neural network based models are here to stay, but as these tools expand so wildly in application, we must begin to ask hard questions about their limitations and implications.

1.1.2 Structure

In this subsection we give a mathematical description of artificial neural networks.

A *neuron* is a nonlinear operator that takes input in $\mathbb{R}^n$ to $\mathbb{R}$, historically designed to emulate the activation characteristics of an organic neuron. A collection of neurons

that are connected via a (usually directed) graph structure are known as an *Artificial Neural Network (ANN).*

The fundamental building blocks of most ANNs are artificial neurons which we will refer to as *perceptrons.*

Definition 1.1.1. *A **perceptron** is a function* $P_{\vec{w}} : \mathbb{R}^n \to \mathbb{R}$ *which has* weights $\vec{w} \in \mathbb{R}^n$ *corresponding with each element of an input vector* $\vec{x} \in \mathbb{R}^n$ *and a bias* $b \in \mathbb{R}$*:*

$$P_{\vec{w}}(\vec{x}) = f\left(\langle \vec{w}, \vec{x} \rangle + b\right)$$

$$P_{\vec{w}}(\vec{x}) = f\left(b + \sum_{i=1}^{n} w_i x_i\right)$$

where $f : \mathbb{R} \to \mathbb{R}$ *is continuous. The function* f *is called the **activation function** for* P*.*

The only nonlinearity in P_w is contained in f. If f is chosen to be linear, then P will be a linear operator. Although this has the advantage of simplicity, linear operators do not perform well on nonlinear problems like classification. For this reason, activation functions are generally chosen to be nonlinear. Historically, heaviside functions were used for activation, later replaced based on work by Malik and Perona (1990) with sigmoids for their smoothness, switching structure, and convenient compactification of the output from each perceptron. It was recently discovered that a simpler nonlinear function, the *Rectified Linear Unit (ReLU)* works as well or better in most neural-network-type applications according to Glorot, Bordes, and Bengio (2011) and additionally training algorithms on ReLU activated networks converge faster according to Nair and Hinton (2010).

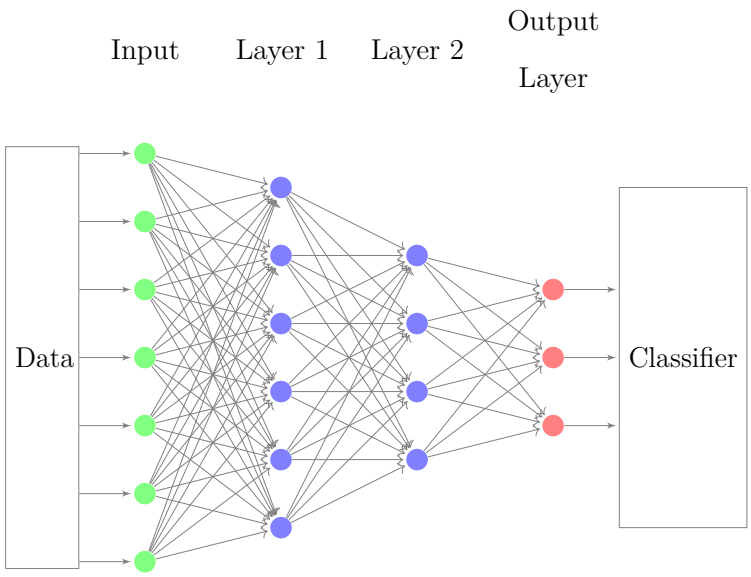
Definition 1.1.2. *The Rectified Linear Unit (ReLU) function is*

$$ReLU(x) = \begin{cases} 0, & x \leq 0; \\ x, & x > 0, \end{cases}$$

Petersen and Voigtlaender (2018) demonstrated that this single nonlinearity of this activation function at $x = 0$ is sufficient to guarantee existence of ϵ approximation of smooth functions by an ANN composed of sufficiently numerous perceptrons connected by ReLU . In addition, ReLU is convex, which enables efficient numerical approximation of smooth functions in shallow networks (Li and Yuan, 2017).

In general ANNs must not be cyclic and, for convenience, are often arranged into independent layers. An early roadblock for neural networks was a proof by Minsky and Papert (1969) that single layers of perceptrons could not encode exclusive-or. Kak (1993) demonstrated that depth, the number of layers in a neural network, is a key factor in its ability to approximate complicated functions including exclusive-or. For this reason, modern ANNs are usually composed of many layers (3-100). The most common instance of a neural network model is a fully connected *feed forward (FF)* configuration. In this configuration data enters as an input layer which is fed into each of the nodes in the first layer of neurons. Output of the first layer is fed into each of the nodes in the second layer, and so on until the output of the final layer is fed into an output filter which generates the final result of the neural network.

In this example of a FF network, an input vector in $\mathbb{R}^7$ is mapped to a an output in $\mathbb{R}^3$ which is fed into a classifier. Each blue circle represents a perceptron with the ReLU activation function.



The output of this ANN is fed into a classifier. To complete this example, we can define the most common classifier, Softmax:

Definition 1.1.3. *Softmax (or the normalized exponential) is the function given by*

$$s : \mathbb{R}^n \to [0,1]^n$$

$$s_j(\vec{x}) = \frac{e^{x_j}}{\sum_{k=1}^{n} e^{x_k}}$$

Definition 1.1.4. *We can define a classifier which picks the class corresponding with the largest output element from Softmax:*

$$\textit{(Output Classification)}\ c_s(\vec{x}) = argmax_i s_i(\vec{x})$$

During training, the output $y \in \mathbb{R}^n$ from a network can thus be compressed using softmax into $[0,1]^n$ as a surrogate for probability for each possible class or directly into

the classes which we can represent as the simplex for the vertices of $[0, 1]^n$ (Bishop, 2006).

Convolutional Neural Networks (CNNs)

Another common type of neural network which is a component in many modern applications including one in the experiments to follow are Convolutional Neural Networks (CNNs). CNNs are fundamentally composed of perceptrons, but each layer is not fully connected to the next. Instead, layers are arranged spatially and overlapping groups of perceptrons are independently connected to the nodes of the next layer, usually with a nonlinear filter that computes the maximum of all of the incoming nodes to a new node. This structure has been shown (e.g. by LeCun, Bengio, et al. (1995)) to be very effective on problems with spatial information.

1.1.3 Training ANNs

Neural networks consist of a very large number of perceptrons with many parameters. Directly solving the system implied by these parameters and the empirical risk minimization problem defined below would be difficult, so we must use a modular approach which takes advantage of the simple and regular structure of ANNs.

A breakthrough came with the application of techniques derived from control theory to ANNs in the late 1980s by Rumelhart, Hinton, and Williams (1986), dubbed backpropagation. This technique was refined into its modern form in the thesis and continuing work of LeCun et al. (1988). In this method, error is propagated backward taking advantage of the directed structure of the network to compute a gradient for each parameter defining it. Because modern ANNs are usually separated into discrete

layers, gradients can be computed in parallel for all perceptrons at the same depth of the network (Bishop, 2006). Leveraging modern GPUs and parallel computing technologies, these gradients can be computed very quickly. There are a number of important considerations in training. We discuss a few in the following subsections.

Selection of the Training Set

The first step in training an ANN is the selection of a training set. ANNs fundamentally are universal function approximators: Given a set of input data and corresponding output data, they approximate a mapping from one to the other. Performance is dependent on how well the phenomenon we hope to model is represented by the training data. The training data must consist of a set of inputs (e.g., images) and a set of outputs (e.g., labels) which contain sufficient examples to characterize the intended model. In a way, this is how we pose a question to the neural network. One must always ask whether the question we wish to pose is well-expressed by the training data we have available.

The most important attributes of a training dataset are the number of samples it contains and its density near where the model will be making predictions. According to conventional wisdom, training a neural network with K parameters will be very challenging if there are fewer than K training samples available. The modular structure of ANNs can be combined with regularization of the weights to overcome these limitations (Liu and Deng, 2015). In general, we will denote a training set by (X, Y) where X is an indexed set of inputs and Y is a corresponding indexed set of labels.

Selecting a Loss Function

Once we have selected a set of training data (both inputs and outputs), we must decide how we will evaluate the match between the ANNs output and the defined outputs from the training dataset – we will quantify the deviation of the ANN compared with the given correspondence as a Loss. In general *loss functions* are nonzero functions which compare an output y against a ground-truth $\hat{y}$. Generally they have the property that an ideal outcome would have a loss of 0.

One commonly used loss function for classification is known as Cross-Entropy Loss:

Definition 1.1.5. *The Cross-Entropy Loss comparing two possible outputs is* $L(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$.

Other commonly used loss functions include L^1 loss (also referred to as Mean Absolute Error (MAE)), L^2 loss (often referred to as Mean-Squared-Error (MSE)), and Hinge Loss (also known as SVM loss).

To set up the optimization, the loss for each training example must be aggregated. Generally, ANN training is conducted via Empirical Risk Minimization where Empirical Risk is defined for a given loss function L as follows:

Definition 1.1.6. *Given a loss function* L*, the Empirical Risk over a training dataset* (X, Y) *of size* N *is*

$$R_{emp}(P_{\vec{w}}(x)) = \frac{1}{N} \sum_{(x,y)\in(X,Y)} L(P_{\vec{w}}(x)), y).$$

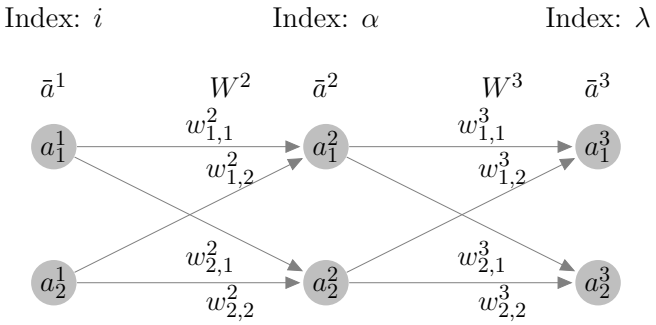
We seek parameters $\vec{w}$ which will minimize $R_{\text{emp}}(P_w(x))$. This will be done with gradient-based optimization.

Computation of Gradient via Backpropagation

Since it is relevant to the optimization being performed, we will briefly discuss the computation of gradients via backpropagation. For this discussion, we will introduce a small subset of a neural network in detail. In general, terms will be indexed as follows:

$$x^{[\text{layer}]}_{[\text{node in layer}],\ [\text{node in previous layer}]}$$

When the second subscript is omitted, the subscript will only index the node in the current layer to which this element belongs.



In this diagram, the W^l are matrices composed of the weights indexed as above. Given an activation function for layer n, A^n and its element-wise application to a vector $\bar{A}^n$. As demonstrated by Krause (2020), we can now write the output $\bar{a}^n$ for any layer of an arbitrary ANN in two ways . Recursively, we can define

$$a^n_\lambda = A^n(\sum_\alpha w^n_{\alpha,\lambda} a^{n-1}_\alpha) \tag{1.1}$$

We can also write the matrix form of this recursion for every node in the layer:

$$\bar{a}^n = \bar{A}^n(W^n(\bar{a}^{n-1})) \tag{1.2}$$

The matrix form makes it easier to write out a closed form for the output of the neural network.

$$\bar{a}^n = \bar{A}^n(W^n(\bar{A}^{n-1}(W^{n-1}(\cdots(\bar{A}^2(W^2\bar{a}^1))\cdots)))) \tag{1.3}$$

Now, given a loss function $L = \sum_i \ell_i(a_i^n)$ where each ℓ_i is a loss function on the i^{th} element of the output, we wish to compute the derivatives $\frac{\partial L}{\partial_{w_{i,j}^l}}$ for every l, i, and j which compose the gradient ∇L. Using the diagram above, we can compute this directly for each weight using chain rule:

$$\frac{\partial L}{\partial w_{\lambda,\alpha}^3} = \frac{\partial L}{\partial a_\lambda^3}\frac{\partial a_\lambda^3}{\partial w_{\lambda,\alpha}^3} = \sum_{\lambda=1}^{n} \ell'_\lambda(a_\lambda^3)(A^3)'(\sum_{\alpha=1}^{n} w_{\alpha,\lambda}^3 a_\alpha^2)a_\alpha^2.$$

Many of the terms of this gradient (e.g. the activations a_i^n and the sums $\sum_i w_{i,j}^n a_i$) are computed during forward propagation when using the network to generate output. We will store such values during the forward pass and use a backward pass to fill in the rest of the gradient. Furthermore, notice that ℓ'_λ and $(A^n)'$ are well understood functions whose derivatives can be computed analytically almost everywhere. We can see that all of the partials will be of the form $\frac{\partial L}{\partial w_{n,i}^l} = \delta_n^l a_i^l$ where δ_n^l will contain terms which are either pre-computed or can be computed analytically. Conveniently, we can define this error signal recursively:

$$\delta_n^l = A'^l(a_n^l)\sum_{i=1}^{n} w_{i,n}^{l+1}\delta_i^{l+1}$$

In matrix form, we have

$$\bar{\delta}^l = \bar{A}'^l(W^l\bar{a}^l) \odot ((W^{l+1})^T\bar{\delta}^{l+1}$$

) Where $\odot$ signifies element-wise multiplication.

Then we can compute the gradient with respect to each layer's matrix W^l as an outer product:

$$\nabla_{W^l} L = \bar{\delta}^l \bar{a}^{(l-1)T}.$$

Since this recursion for layer n only requires information from layer $n+1$, this allows us to propagate the error signals that we compute backwards through the network.

Optimization of Weights

Given a set of training input data and a method for computing gradients, our ultimate goal is to iteratively run our training-data through the network, updating weights gradually according to the gradients computed by backpropagation. In general, we start with some default arrangement of the weights and choose a step size η for gradient descent. Then for each weight, in each iteration of the learning algorithm, we apply a correction so that

$$w'_{i',j',k'} = w_{i',j',k'} - \eta \frac{\partial E(Y, \hat{Y})}{\partial w_{i',j',k'}}$$

In this case, the step size (learning rate) η is fixed throughout training. Numerical computation of the gradient requires first evaluating the network forward by computing the output for a given input. The value of every node in the network is saved and these values are used to weight the error as it is propagated backward through the network. Once the gradient is computed, the weights are adjusted according to the step defined above. This process is repeated until convergence is attained to within a tolerance. It should be clear from the number of terms in this calculation that the

initial guess and step size can have significant effect on the eventual trained weights. Due to lack of a guarantee for general convexity, Bishop (2006) observed that poor guesses for such a large number of parameters can lead to gradients blowing up or down. Due to non-linearity and the plenitude of local minima in the loss function, classic gradient descent usually does not perform well during ANN training.
By far the most common technique for training the weights of neural networks adds noise in the form of random re-orderings of the training data to the general optimization process and is known as stochastic gradient descent.

Definition 1.1.7. *Stochastic Gradient Descent (SGD)*

Given an ANN $N : \mathbb{R}^n \to C$*, an initial set of weights for this network* $\vec{w}_0$ *(usually a small random perturbation from 0), a set of training data* X *with labels* Y*, and a learning rate* η*, the algorithm is as follows:*

Batch Stochastic Gradient Descent

$w = w_0$
while $E(\hat{Y}, P_w(X))$ (cumulative loss) is still improving **do** ▷ (the stopping condition may require that the weight change by less than ε for some number of iterations or could be a fixed number of steps)
  Randomly shuffle (X, Y)
  Draw a small batch $(\hat{X}, \hat{Y}) \subset (X, Y)$
  $w \leftarrow w - \eta \left(\sum_{(x,y)\in(\hat{X},\hat{Y})} \nabla L(P_w(\hat{x}), \hat{y}) \right)$
end while

Stochastic gradient descent achieves a smoothing effect on the gradient optimization by only sampling a subset of the training data for each iteration. Miraculously, this smoothing effect not only often achieves faster convergence, the result also generalizes better than solutions using deterministic gradient methods (Hardt, Recht, and Singer, 2015). It is for this reason that SGD has been adopted as the de facto standard among ANN training applications.

Chapter 2 Adversarial Attacks

Deep Neural Networks (DNNs) and their variants are core to the success of modern machine learning as summarized by Prakash et al. (2018). They have dominated competitions in image processing, optical character recognition, object detection, video classification, natural language processing, and many other fields (Schmidhuber, 2015). Ten years ago an interesting property of such networks was observed by Szegedy et al. (2014). Their approach was to define a loss function relating the output of the ANN for a given initial image to a target adversarial output plus the L^2-norm of the input and use backpropagation to compute gradients – not on the weights of the neural network, but on just the input layer to the network. The solution to this optimization problem, efficiently approximated by a gradient-based optimizer, would be a slightly perturbed natural input with a highly perturbed output. Their experimental results are striking, which we can see in Figure. 2.1. More mysteriously, these examples are often transferable – an attack generated against one model may succeed against a totally different model. With the incredible expansion of the application of universal function approximators in machine-learning, their reliability has come to have real-world significance. Self-driving cars, including those manufactured by Tesla Incorporated, use image classification models to distinguish stop-signs and speed-limit signs. Evtimov et al. (2017) have shown that these models are not robust!. Other machine-learning (ML) models are increasingly relied upon by the defense intelligence apparatus (Hutchins, Cloppert, Amin, et al., 2011). Social media and search engines which are now the backbone of the internet use ML increasingly to determine what

content will receive attention. In order to wisely use these tools, it is crucial that we carefully understand their limitations.

Adversarial examples occur when natural data can be perturbed in small ways in order to produce a similar input which receives a significantly different model output. "Small" in this context may refer to small in a particular metric or sometimes is referred to in the context of human perception. It is important to note that Adversarial examples are not just a peculiarity, but seem to occur for most, if not all, ANN classifiers. For example, Shafahi et al. (2018) used isoperimetric inequalities on high dimensional spheres and hypercubes to conclude that there is a reasonably high probability that each correctly classified data point has a nearby adversarial example. Ilyas et al. (2019) argued that optimized models use some subtle features for classification which are neither intuitive to humans nor robust to perturbation. The argue that ML models can efficiently extract features from training data, but that they do not connect these features robustly across scales. The prevalence of these features is illustrated by Madry et al. (2018) with the simple experiment of adding vast quantities of adversarially perturbed data during training. Although this method increases adversarial robustness at a cost to prediction accuracy (Tsipras et al., 2018), it does not do so very significantly, and leaves behind vulnerabilities that can still be reduced to non-robust features (Shafahi et al., 2018).

We will take a geometric approach to analyzing robustness, both in terms of the models' understanding of underlying data geometry and by carefully defining the decision boundary of a model and studying its properties. There have been many attempts to identify adversarial examples using properties of the decision boundary. Fawzi et al. (2018) found that decision boundaries tend to have highly curved regions,

and these regions tend to favor negative curvature, indicating that regions that define classes are highly nonconvex. These were found for a variety of ANNs and classification tasks. A related idea is that adversarial examples often arise within cones, outside of which images are classified in the original class, as observed by Roth, Kilcher, and Hofmann (2019). Many theoretical models of adversarial examples, for instance the dimple model developed by Shamir (2021), have high curvature and/or sharp corners as an essential piece of why adversarial examples can exists very close to natural examples.

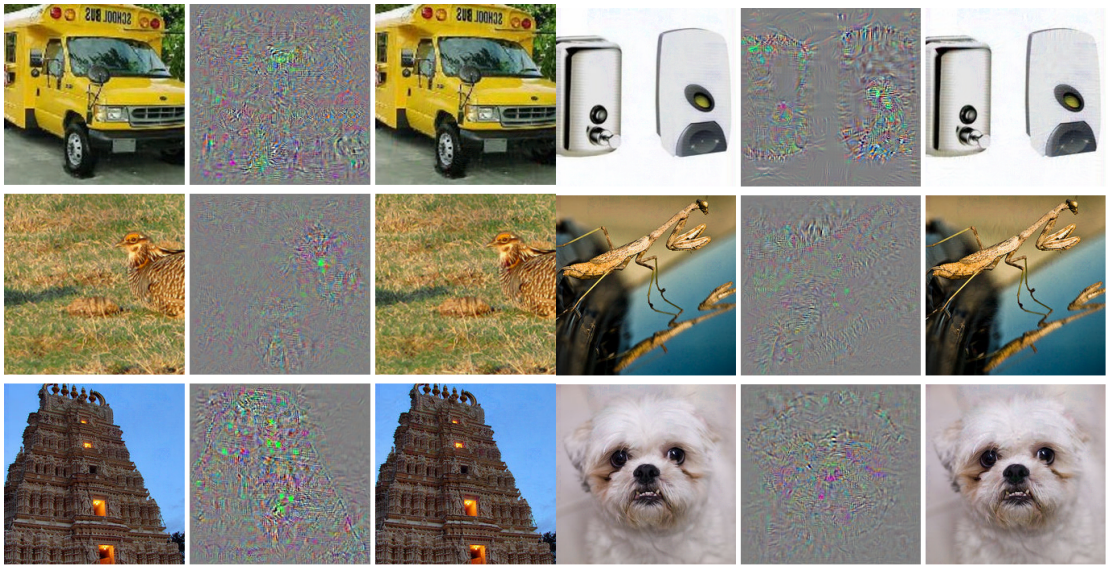


FIGURE 2.1: Natural Images are in columns 1 and 4, Adversarial images are in columns 3 and 6, and the difference between them (magnified by a factor of 10) is in columns 2 and 5. All images in columns 3 and 6 are classified by AlexNet as "Ostrich" (Szegedy et al., 2014).

2.1 Common Datasets

The first step toward understand adversarial attacks understanding the data on which neural networks are built. We will limit our investigation mostly to classic image classification problems, although several of our results will hold more generally. The data set used above in Figure. 2.1 is known as ImageNet – a large set of labeled images varying in size originally compiled for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC Russakovsky et al. (2015)). This dataset has become a standard for image classification and feature identification experiments. In the experiments that follow, ImageNet will be featured alongside the Modified National Institute of Standards and Technology dataset (MNIST LeCun and Cortes (2010)) which is a database of hand written digits often used to develop image processing and character recognition systems. This dataset is much lower resolution than ImageNet and therefore experiments run much more quickly on it and require less complex input/output.

2.2 Common Attack Techniques

Adversarial attacks are generally produced by introducing an objective function. This objective balances achieving a change in predicted classification with minimizing the perturbation needed to achieve the desired prediction. The adversarial objective can use *cross-entropy loss* to compare predictions against a specific target or the negation of the original model prediction for a given input (Good, 1963). Perturbation size is often measured using a regularization term in image space (e.g. the L^2 norm) which penalizes the generated adversary for being too far from its starting point. This loss

function is combined with an optimization algorithm in order to produce an attack technique.

2.2.1 L-BFGS minimizing distortion

The original attack used by Szegedy et al. (2014) set up a box-constrained optimization problem whose approximated solution generates these targeted misclassifications. We will write this precisely according to their formulation:

Let $f : \mathbb{R}^m \to \{1, ..., k\}$ be a classifier and assume f has an associated continuous loss function denoted by $\text{loss}_f : \mathbb{R}^m \times \{1, ..., k\} \to \mathbb{R}^+$ and l a target adversarial class or output.

Minimize $\|r\|_2$ subject to:

1. $f(x + r) = l$
2. $x + r \in [0, 1]^m$

The solution is approximated with L-BFGS (see Appendix A.1) as implemented in Pytorch or Keras. This technique yields examples that are close to their original counterparts in the L^2 sense, but are predicted to be another class by the model with high confidence.

L-BFGS: Mnist The following examples are prepared by implementing the above technique via pytorch (Paszke et al., 2019) on images from the Mnist dataset with FC200-200-10, a neural network with 2 hidden layers with 200 nodes each in Figure. 2.2: Szegedy et al. define a metric to compare the magnitude of these perturbations:

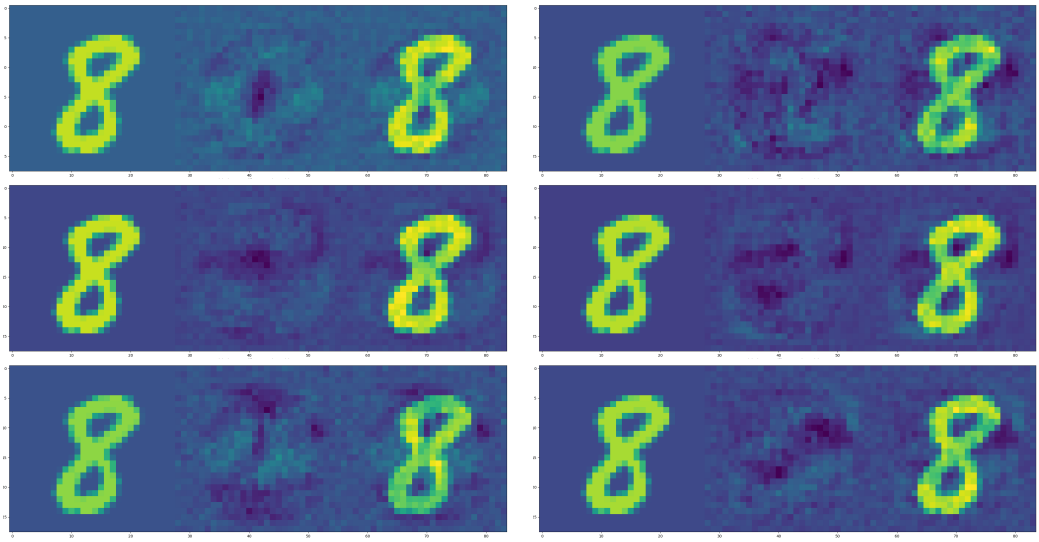


FIGURE 2.2: Original images on the left, Perturbation is in the middle, Adversarial Image (total of Original with Perturbation) is on the right. Column 1 shows an original 8 being perturbed to adversarial classes 0, 2, and 4. Column 2 shows adversarial classes 1, 3, and 5

Definition 2.2.1. *Distortion is the L^2 norm of the difference between an original image and a perturbed image, divided by the square root of the number of pixels in the image:*

$$\sqrt{\frac{\sum_i (\hat{x}_i - x_i)^2}{n}}$$

Distortion is L^2 magnitude normalized by the square-root of the number of dimensions so that values can be compared for modeling problems with differing numbers of pixels.

900 examples were generated for the network above. We measured an average distortion of 0.089 with a distribution given in Figure. 2.3. Another histogram is provided for distortions measured from attacks against the VGG16 (Visual Geometry Group Network 16) network trained on the ImageNet dataset in. This histogram demonstrates that ImageNet networks are vulnerable to much more subtle adversarial

attacks.

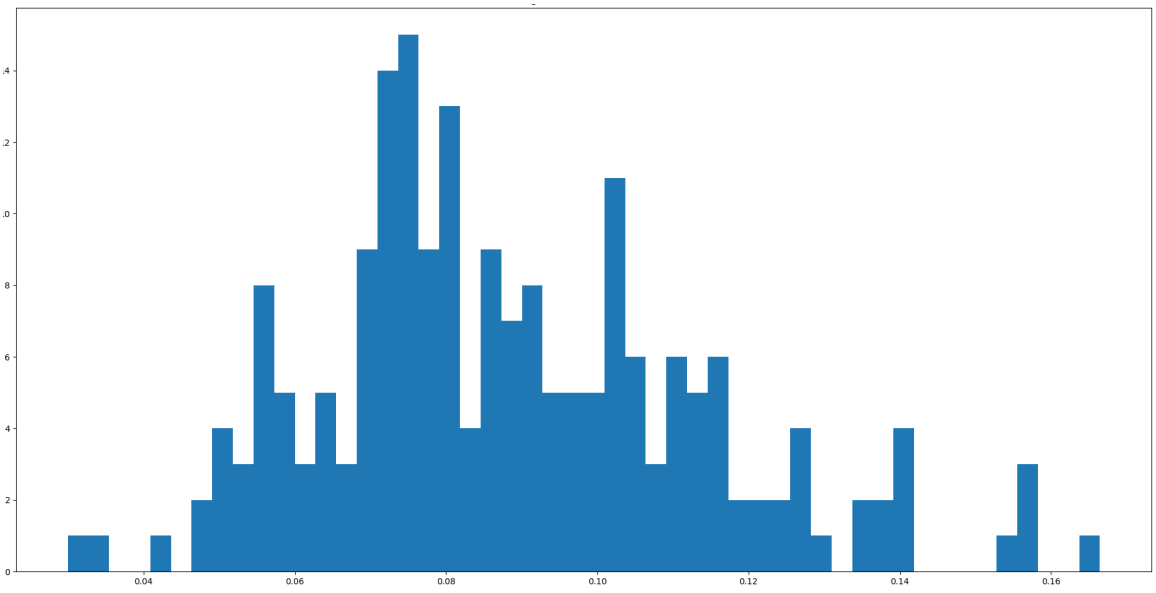


FIGURE 2.3: A histogram of the distortion measured for each of 900 adversarial examples generated using L-BFGS against the FC-200-200-10 network on Mnist. Mean distortion is 0.089.

L-BFGS: ImageNet We also tried to replicate the results of Szegedy et al. (2014) on ImageNet. Attacking VGG16, a well known model from the ILSVRC-2014 competition (Simonyan and Zisserman, 2014), on ImageNet images with the same technique generates the examples in Figure. 2.4:

Fast Gradient Sign Method (FGSM) As the study of adversarial examples has expanded, it has become known that often very simple single-step attacks are successful and sufficiently subtle. Goodfellow, Shlens, and Szegedy (2014) proposed one such attack which we have also implemented. This is a single step attack process which uses the sign of the gradient of the loss function L with respect to the image to

FIGURE 2.4: Original images on the left, Perturbation (magnified by a factor of 100) by is in the middle, Adversarial Image (total of Original with Perturbation) is on the right.

find the adversarial perturbation. For given ε, the modified image $\hat{x}$ is computed as

$$\hat{x} = x + \epsilon \text{sign}(\nabla L(P_w(x), x)) \tag{2.1}$$

This method is simpler and much faster to compute than the L-BFGS technique described above, but produces adversarial examples less reliably and with generally larger distortion. Performance was similar but inferior to the Iterative Gradient Sign Method summarized below.

Iterative Gradient Sign Method (IGSM) In work by Kurakin, Goodfellow, and Bengio (2016) an iterative application of FGSM was proposed. After each iteration, the image is clipped to a εL_∞ neighborhood of the original. Let $x'_0 = x$, then after m iterations, the adversarial image obtained is:

$$x'_{m+1} = \text{Clip}_{x,\epsilon}\Big\{x'_m + \alpha \times \text{sign}(\nabla \ell(F(x'_m), x'_m))\Big\} \tag{2.2}$$

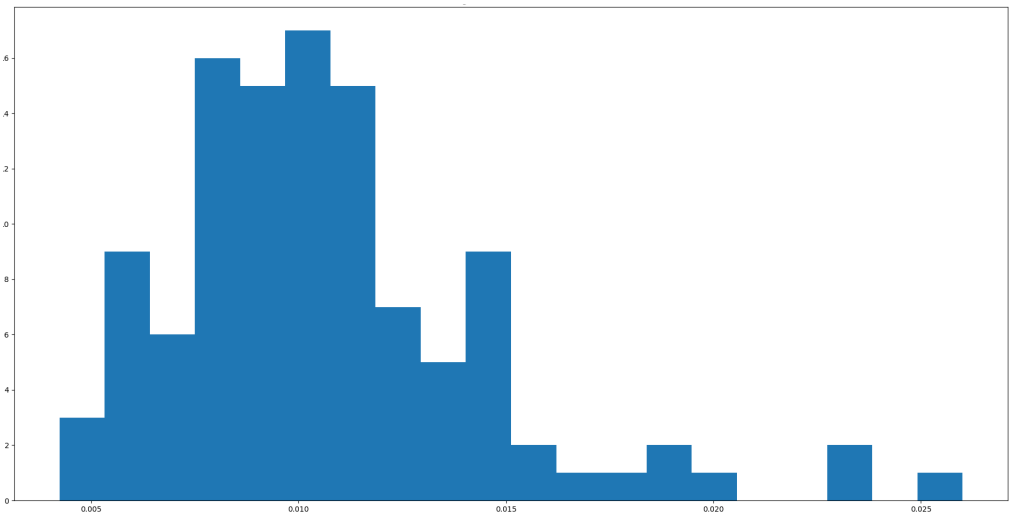


FIGURE 2.5: A histogram of the distortion measured for each of 112 adversarial examples generated using L-BFGS against the VGG16 network on ImageNet images with mean distortion 0.0107

Where $\text{Clip}_{x,\varepsilon}$ takes the minimum of $x + \varepsilon$ and x'_m for elements larger than $x + \varepsilon$ and vice versa. This method is faster than L-BFGS and more reliable than FGSM but still produces examples with greater distortion than L-BFGS. An example is shown in Figure. 2.6.

2.2.2 Other Attacks

The following attack techniques are also prevalent in the literature

Jacobian-based Saliency Map Attack (JSMA) Another attack noted by Papernot et al. (2015) estimates the *saliency map*, a rating for each of the input features (e.g. each pixel) on how influential it is for causing the model to predict a particular class with respect to the model output (Wiyatno and Xu, 2018). This attack modifies the pixels that are most salient. This is a targeted attack, and saliency is designed to find

the pixel which increases the classifier's output for the target class while tending to decrease the output for other classes.

Deep Fool (DFool) A technique proposed by Moosavi-Dezfooli, Fawzi, and Frossard (2015) to generate an un-targeted iterative attack. This method approximates the classifier as a linear decision boundary and then finds the smallest perturbation needed to cross that boundary. This attack minimizes L_2 norm with respect to to the original image.

Carlini & Wagner (C&W) In work by Carlini and Wagner (2016) an adversarial attack is proposed which updates the loss function such that it jointly minimizes L^p and a custom differentiable loss function based on un-normalized outputs of the classifier (called *logits*). Let Z_k denote the logits of a model for a given class k, and κ a margin parameter. Then C&W tries to minimize:

$$||x - \hat{x}||_p + c * max\left(Z_k(\hat{x}_y) - max\{Z_k(\hat{x}) : k \neq y\}, -\kappa\right) \tag{2.3}$$

2.2.3 Attack Standards and Toolbox

Since adversarial robustness has expanded as a field, many papers have been released pushing various methods for defending against adversarial attacks. While initially this approach – producing a defense that fit a narrow context and releasing it to the community for evaluation was seen as useful. However, most such approaches would inevitably face simple rebuttals by small modification of the attack techniques used. Carlini and their group gained a particular reputation for brief rebuttals (Carlini and Wagner, 2016; Papernot et al., 2016) of such methods. These approaches were

finally codified by Tramèr et al. ("On Adaptive Attacks to Adversarial Example Defenses") in the form of a set of guidelines that should be used to attack any proposed defense before releasing it to the community. This high bar has greatly reduced the number of low quality defenses which gain attention, but it has also demonstrated the incredible difficulty of producing successful general defenses against adversarial attacks. Despite its poor performance, the strategy of adversarial training proposed by Tramer and Boneh (2019) is one of the few defenses which have maintained any advantage under the Tramer/Carlini adaptive framework.

2.3 Theory of Adversarial Examples

Despite the prevalence of studies developing and analyzing adversarial attacks, the field is characterized by a plethora of definitions for what it means to be "adversarial". We will analyze a few of these in order to develop our own precise definitions. Indeed, defining an adversarial example is intimately related with the task of identification, which leaves a paradox of sorts: If we can precisely define an adversarial example and that definition allows us to identify them, then that definition constitutes a perfect defense. In practice, however, we know this is at least not trivial.

2.3.1 Defining Adversarial Attacks

Roth, Kilcher, and Hofmann (2019) proposed a statistical method to identify adversarial examples from natural data. Their main idea was to consider how the last layer in the neural network (the logit layer) would behave on small perturbations of a natural example. This is then compared to the behavior of a potential adversarial

example. If it differs by a predetermined threshold, the example is flagged as adversarial. Successfully flagging adversarial examples in this way works best when adversarial examples tend to perturb toward the original class from which the adversarial example was perturbed. However, this is not always the case. It was shown by Hosseini, Kannan, and Poovendran (2019) that it is possible to produce adversarial examples, for instance using a logit mimicry attack, that instead of perturbing an adversarial example toward the true class, actually perturb to some other background class. In fact, we will see in Section 3.4.1 that the emergence of a background class, which was observed as well by Roth, Kilcher, and Hofmann (2019), is quite common.

We primarily consider adversarial examples for classifiers. Let X be a set of possible data and let L be a set of labels. We will consider classifier as a map $\mathcal{C} : X \to L$. In general X may be much larger than the actual space from which our data are drawn. If the data actually come from a submanifold of X, we call this the *data submanifold*. The data submanifold may not be a strict submanifold, and we often do not know the shape or even dimension of it.

Data is drawn from a distribution μ on X that is usually not known. The overarching goal of classification is to produce a classifier such that $\mathcal{C}$ is as good as possible on the support of μ. We define $X_N \subseteq X$ to be the support of μ and call it the set of *natural data*. Usually our classification problem is the following: given a set of i.i.d. samples $\Sigma \sim \mu$, where we consider $\Sigma \subseteq X_N$, and a classifier $\mathcal{C}_\Sigma$ on Σ, find a classifier $\mathcal{C}$ on X such that $\mathcal{C}$ lies in some class of "good functions" in such a way that it is relatively good at interpolating and/or extrapolating $\mathcal{C}_\Sigma$. In particular, we hope that $\mathcal{C}$ is as accurate as possible on the support of μ, which we call the *natural data*. The classifier $\mathcal{C}$ partitions X into classes, each of which is defined as $\mathcal{C}^{-1}(\ell)$ for some

$\ell \in L$. Points on the boundaries of these classes do not have a clear choice of label, and the points in X on the boundaries of the classes make up the *decision boundary* for $\mathcal{C}$.

To build up to a mathematical framework for adversarial attacks in the context of geometric analysis, we develop definitions and terms to refer to adversarial examples without relying on subjective characteristics like human vision. Let X denote a set of possible data and L denote a set of labels that distinguish the different classes. We are now ready to define adversarial examples.

Definition 2.3.1. *Let d be a metric on X, let $x \in X$ have label $\ell \in L$, and let $\mathcal{C} : X \to L$ be a classifier. We say that x admits an* (ε, d)–adversarial example *to $\mathcal{C}$ if there exists $\hat{x} \in X$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) \neq \ell$. We say that x admits an* $(\varepsilon, d)-$adversarial example *on $\mathcal{C}$ if there exists a point $\hat{x}$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) \neq \ell$.*

One typically considers Definition 2.3.1 in the context of small ε. Often consideration is made of when such a misclassification is a result of an intentional act by an adversary. There are various methods of producing adversarial examples which are discussed later. In some cases, the adversarial label is explicitly targeted:

Definition 2.3.2. *Let d be a divergence on X, let $x \in X$ have label $\ell \in L$, and let $\mathcal{C} : X \to L$ be a classifier. Let $\varepsilon > 0$ and $\ell_t \neq \ell$ be fixed. We say that x admits an* (ε, d, ℓ_t)–targeted adversarial example *to $\mathcal{C}$ if there exists $\hat{x} \in X$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) = \ell_t$. Consider a point $x \in X$ with corresponding class $\ell \in C$ and a classifier $\mathcal{C} : X \to C$. We say that x admits an* $(\varepsilon, d, \ell_t)-$targeted adversarial example *if there exists a point $\hat{x}$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) = \ell_t$.*

These definitions rely on a metric d, emphasizing the reliance on the choice of distance to understand notions of closeness. From here on, we will assume that (X, d) is a Euclidean vector space with d being the Euclidean metric. This will allow for the use of standard Gaussian distributions as well.

The solution to this optimization problem, efficiently approximated by a gradient-based optimizer, would be a slightly perturbed natural input with a highly perturbed output. We have already shown several examples of these techniques being applied, and one more example can be seen in Figure. 2.7 for AlexNet, a cutting-edge model designed by Krizhevsky, Sutskever, and Hinton (2012) in collaboration with Geoffrey Hinton. [1] There has since been significant work describing methods of producing and identifying adversarial examples. In the next chapter, we will attempt to develop a framework to understand what properties of an ANN are being exploited by such methods.

[1]Note that Alex Stutskever was both a member of DNNResearch, Hinton's company which was acquired by Google to become the core of Google Brain and since 2015 he has been Chief Scientist at OpenAI. He was in the news recently as a member of the board of OpenAI in November 2023 when Sam Altman was fired as CEO and then re-hired a week later.

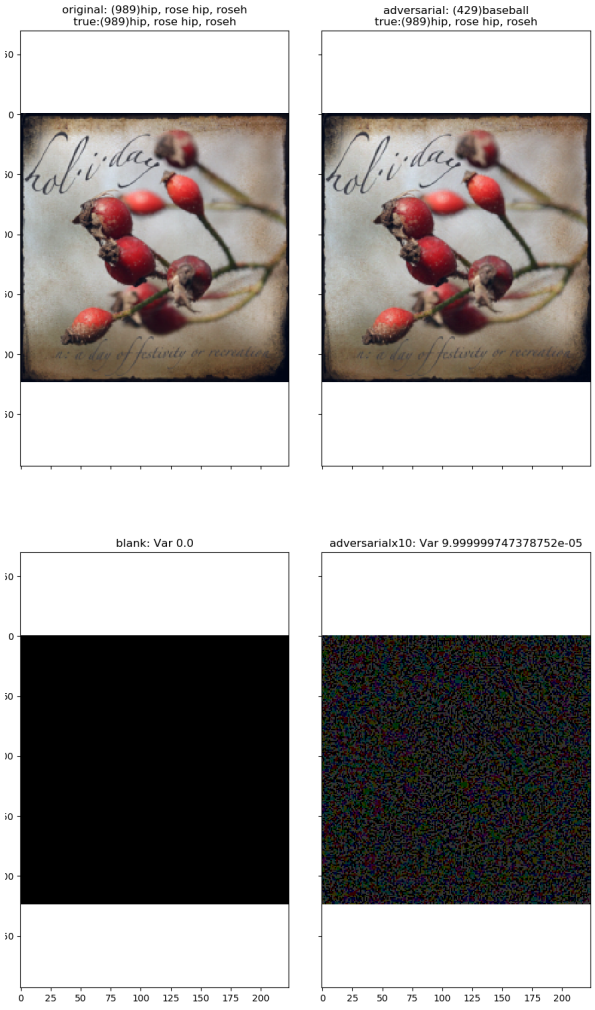


FIGURE 2.6: adversarial example generated against VGG16 (ImageNet) with IGSM. Original Image on the left, adversarial image and added noise (ratio of variance adversarial noise/original image: 0.0000999) on the right.

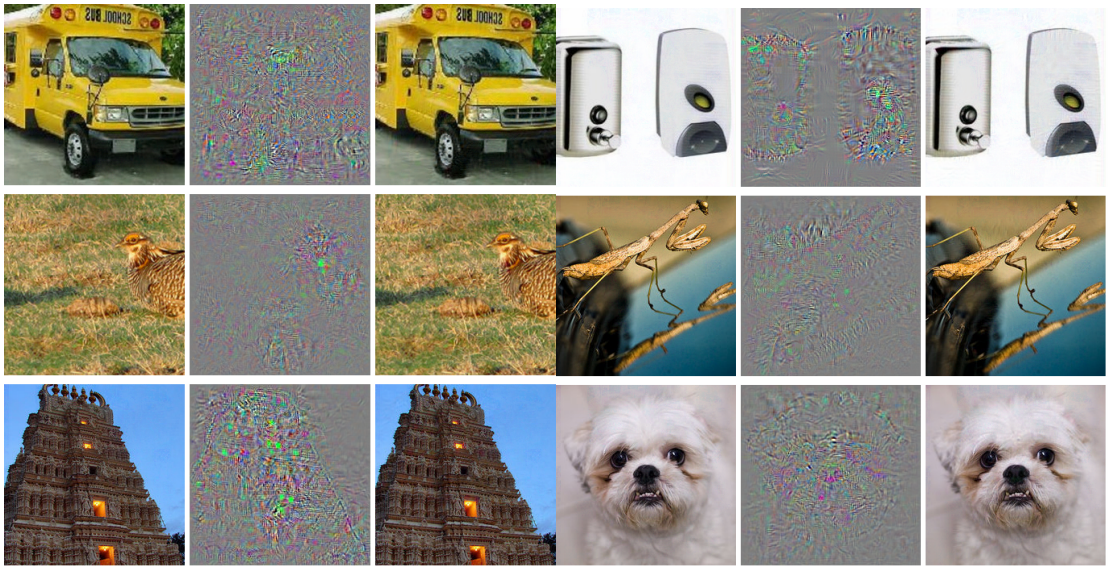


FIGURE 2.7: Natural Images are in columns 1 and 4, Adversarial images are in columns 3 and 6, and the difference between them (magnified by a factor of 10) is in columns 2 and 5. All images in columns 3 and 6 are classified by AlexNet as "Ostrich" (Szegedy et al., 2014)

Chapter 3 Persistent Classification

This chapter is devoted to defining geometric properties that can be used to assess robustness of neural networks and to define adversarial examples in testable and separable ways. This synthesizes the concept of Persistence, a distributional method for analyzing curvature around data, with the optimization tasks used to generate adversarial examples and the direct measurement of properties of decision boundaries for neural networks. The high angles of incidence observed when interpolating across decision boundaries of neural networks among natural images and the much higher angles observed when interpolating among adversarial examples indicate that geometric properties may explain or be related to why adversarial examples are so easy to find for certain networks. This paper is in submission to the CODA Journal, and is composed of theoretical and practical work on Persistence and decision boundary incidence angles and other properties all conducted by Brian Bell. A second body of work analyzing the manifold alignment of classification models is related to this work, attempting to correct some of these geometric inadequacies by forcing networks to be orthogonal or parallel in appropriate regimes to approximated manifolds in specific regimes. This second body of work was completed by Michael Geyer.

Whereas Roth, Kilcher, and Hofmann (2019) consider adding various types of noise to a given point and Hosseini, Kannan, and Poovendran (2019) consider small Gaussian perturbations of x sampled from $N(x, \varepsilon^2 I)$ for small ε, we specifically focus on tuning the standard deviation parameter to determine a statistic describing how a given data point is placed within its class. The γ-persistence then gives a measurement

similar to distance to the boundary but that is drawn from sampling instead of distance. The sampling allows for a better description of the local geometry of the class and decision boundary, as we will see in Section 3.3.1. Our statistic is based on the fraction of a Gaussian sampling of the neighborhood of a point that receives the same classification; this is different from that of Roth, Kilcher, and Hofmann (2019), which is the expected difference of the output of the logit layer of the original data point and the output of the logit layer for perturbed data points. Additionally, while their statistics are defined pairwise with reference to pre-chosen original and candidate classes, ours is not.

There are a number of hypotheses underlying the existence of adversarial examples for classification problems, including the high-dimensionality of the data, high codimension in the ambient space of the data manifolds of interest, and that the structure of machine learning models may encourage classifiers to develop decision boundaries close to data points. This article proposes a new framework for studying adversarial examples that does not depend directly on the distance to the decision boundary. Similarly to the smoothed classifier literature, we define a (natural or adversarial) data point to be (γ, σ)-stable if the probability of the same classification is at least γ for points sampled in a Gaussian neighborhood of the point with a given standard deviation σ. We focus on studying the differences between persistence metrics along interpolants of natural and adversarial points. We show that adversarial examples have significantly lower persistence than natural examples for large neural networks in the context of the MNIST and ImageNet datasets. We connect this lack of persistence with decision boundary geometry by measuring angles of interpolants with respect to decision boundaries. Finally, we connect this approach with robustness by developing

a manifold alignment gradient metric and demonstrating the increase in robustness that can be achieved when training with the addition of this metric.

3.1 Introduction

Deep Neural Networks (DNNs) and their variants are core to the success of modern machine learning (Prakash et al., 2018), and have dominated competitions in image processing, optical character recognition, object detection, video classification, natural language processing, and many other fields (Schmidhuber, 2015). Yet such classifiers are notoriously susceptible to manipulation via adversarial examples (Szegedy et al., 2014). Adversarial examples occur when natural data can be subject to subtle perturbation which results in substantial changes in output. Adversarial examples are not just a peculiarity, but seem to occur for most, if not all, DNN classifiers. For example, Shafahi et al. (2018) used isoperimetric inequalities on high dimensional spheres and hypercubes to conclude that there is a reasonably high probability that a correctly classified data point has a nearby adversarial example. This has been reiterated using mixed integer linear programs to rigorously check minimum distances necessary to achieve adversarial conditions (Tjeng, Xiao, and Tedrake, 2017). Ilyas et al. (2019) showed that adversarial examples can arise from features that are good for classification but not robust to perturbation.

There have been many attempts to identify adversarial examples using properties of the decision boundary. Fawzi et al. (2018) found that decision boundaries tend to have highly curved regions, and these regions tend to favor negative curvature, indicating that regions that define classes are highly nonconvex. The purpose of this work is to

investigate these geometric properties related to the decision boundaries. We will do this by proposing a notion of stability that is more nuanced than simply measuring distance to the decision boundary, and is also capable of elucidating information about the curvature of the nearby decision boundary. We develop a statistic extending prior work on smoothed classifiers by Cohen, Rosenfeld, and Kolter (2019). We denote this metric as Persistence and use it as a measure of how far away from a point one can go via Gaussian sampling and still consistently find points with the same classification. One advantage of this statistic is that it is easily estimated by sending a Monte Carlo sampling about the point through the classifier. In combination with this metric, direct measurement of decision boundary incidence angle with dataset interpolation and manifold alignment can begin to complete the picture for how decision boundary properties are related with neural network robustness.

These geometric properties are related to the alignment of gradients with human perception (Ganz, Kawar, and Elad, 2022; Kaur, Cohen, and Lipton, 2019; Shah, Jain, and Netrapalli, 2021) and with the related underlying manifold (Kaur, Cohen, and Lipton, 2019; Ilyas et al., 2019) which may imply robustness. For our purposes, Manifold Aligned Gradients (MAG) will refer to the property that the gradients of a model with respect to model inputs follow a given data manifold $\mathcal{M}$ extending similar relationships from other work by Shamir, Melamed, and BenShmuel (2021).

Contributions. We believe these geometric properties are related to why smoothing methods have been useful in robustness tasks (Cohen, Rosenfeld, and Kolter, 2019; Lecuyer et al., 2019; Li et al., 2019). We propose three approaches in order to connect robustness with geometric properties of the decision boundary learned by DNNs:

1. We propose and implement two metrics based on the success of smoothed

classification techniques: (γ, σ)-stability and γ-persistence defined with reference to a classifier and a given point (which can be either a natural or adversarial image, for example) and demonstrate their validity for analyzing adversarial examples.

2. We interpolate across decision boundaries using our persistence metric to demonstrate an inconsistency at the crossing of a decision boundary when interpolating from natural to adversarial examples.

3. We demonstrate via direct interpolation across decision boundaries and measurement of angles of interpolating vectors relative to the decision boundary itself that dimensionality is not solely responsible for geometric vulnerability of neural networks to adversarial attack.

3.2 Motivation and related work

Our work is intended to shed light on the existence and prevalence of adversarial examples to DNN classifiers. It is closely related to other attempts to characterize robustness to adversarial perturbations, and here we give a detailed comparison.

Distance-based robustness.

A typical approach to robustness of a classifier is to consider distances from the data manifold to the decision boundary (Wang et al., 2020; Xu et al., 2023b; He, Li, and Song, 2018). Khoury and Hadfield-Menell (2018) define a classifier to be robust if the class of each point in the data manifold is contained in a sufficiently large ball that is entirely contained in the same class. The larger the balls, the more robust the classifier. It is then shown that if training sets are sufficiently dense in

relation to the reach of the decision axis, the classifier will be robust in the sense that it classifies nearby points correctly. In practice, we do not know that the data is so well-positioned, and it is quite possible, especially in high dimensions, that the reach is extremely small, as evidenced by results on the prevalence of adversarial examples, e.g., Shafahi et al. (2018) and in evaluation of ReLU networks with mixed integer linear programming e.g., Tjeng, Xiao, and Tedrake (2017).

Tsipras et al. (2018) investigated robustness in terms of how small perturbations affect the the average loss of a classifier. They define standard accuracy of a classifier in terms of how often it classifies correctly, and robust accuracy in terms of how often an adversarially perturbed example classifies correctly. It was shown that sometimes accuracy of a classifier can result in poor robust accuracy. Gilmer et al. (2018a) use the expected distance to the nearest different class (when drawing a data point from the data distribution) to capture robustness, and then show that an accurate classifier can result in a small distance to the nearest different class in high dimensions when the data is drawn from concentric spheres. May recent works (He, Li, and Song, 2018; Chen et al., 2023; Jin et al., 2022) have linked robustness with decision boundary dynamics, both by augmenting training with data near decision boundaries, or with dynamics related to distances from decision boundaries. We acknowledge the validity of this work, but will address some of its primary limitations by carefully studying the orientation of the decision boundary relative to model data.

A related idea is that adversarial examples often arise within cones, outside of which images are classified in the original class, as observed by Roth, Kilcher, and Hofmann (2019). Many theoretical models of adversarial examples, for instance the dimple model developed by Shamir (2021), have high curvature and/or sharp corners

as an essential piece of why adversarial examples can exists very close to natural examples.

Adversarial detection via sampling. While adversarial examples often occur, they still may be rare in the sense that most perturbations do not produce adversarial examples. Hu et al. (2019) used the observation that adversarial examples are both rare and close to the decision boundary to detect adversarial examples. They take a potential data point and look to see if nearby data points are classified differently than the original data point after only a few iterations of a gradient descent algorithm. If this is true, the data point is likely natural and if not, it is likely adversarial. This method has been generalized with the developing of smoothed classification methods (Cohen, Rosenfeld, and Kolter, 2019; Lecuyer et al., 2019; Li et al., 2019) which at varying stages of evaluation add noise to the effect of smoothing output and identifying adversaries due to their higher sensitifity to perturbation.. These methods suffer from significant computational complexity (Kumar et al., 2020) and have been shown to have fundamental limitations in their ability to rigorously certify robustness (Blum et al., 2020; Yang et al., 2020a). We will generalize this approach into a metric which will allow us to directly study these limitations in order to better understand how geometric properties have given rise to adversarial vulnerabilities. In general, the results of Hu et al. (2019) indicate that considering samples of nearby points, which approximate the computation of integrals, is likely to be more successful than methods that consider only distance to the decision boundary.

Roth, Kilcher, and Hofmann (2019) proposed a statistical method to identify adversarial examples from natural data. Their main idea was to consider how the last layer in the neural network (the logit layer) would behave on small perturbations of a

natural example. This is then compared to the behavior of a potential adversarial example.

It was shown by Hosseini, Kannan, and Poovendran (2019) that it is possible to produce adversarial examples, for instance using a logit mimicry attack, that instead of perturbing an adversarial example toward the true class, actually perturb to some other background class. In fact, we will see in Section 3.4.1 that the emergence of a background class, which was observed as well by Roth, Kilcher, and Hofmann (2019), is quite common. Although many recent approaches have taken advantage of these facts (Taori et al., 2020; Lu et al., 2022; Osada et al., 2023; Blau et al., 2023) in order to measure and increase robustness, we will leverage these sampling properties to develop a metric directly on decision-boundary dynamics and how they relate to the success of smoothing based robustness.

Manifold Aware Robustness

The sensitivity of convolutional neural networks to imperceptible changes in input has thrown into question the true generalization of these models. Jo and Bengio (2017) study the generalization performance of CNNs by transforming natural image statistics. Similarly to our MAG approach, they create a new dataset with well-known properties to allow the testing of their hypothesis. They show that CNNs focus on high level image statistics rather than human perceptible features. This problem is made worse by the fact that many saliency methods fail basic sanity checks (Adebayo et al., 2018; Kindermans et al., 2019).

Until recently, it was unclear whether robustness and manifold alignment were directly linked, as the only method to achieve manifold alignment was adversarial training. Along with the discovery that smoothed classifiers are perceptually aligned,

comes the hypothesis that robust models in general share this property put forward by Kaur, Cohen, and Lipton (2019). This discovery raises the question of whether this relationship between manifold alignment of model gradients and robustness is bidirectional.

Khoury and Hadfield-Menell (2018) study the geometry of natural images, and create a lower bound for the number of data points required to effectively capture variation in the manifold. Unfortunately, they demonstrate that this lower bound is so large as to be intractable. Shamir, Melamed, and BenShmuel (2021) propose using the tangent space of a generative model as an estimation of this manifold. Magai and Ayzenberg (2022) thoroughly review certain topological properties to demonstrate that neural networks intrinsically use relatively few dimensions of variation during training and evaluation. Vardi, Yehudai, and Shamir (2022) demonstrate that even models that satisfy strong conditions related to max margin classifiers are implicitly non-robust. PCA and manifold metrics have been recently used to identify adversarial examples (Aparne, Banburski, and Poggio, 2022; Nguyen Minh and Luu, 2022). We will extend this work to study the relationship between robustness and manifold alignment directly by baking alignment directly into networks and comparing them with another approach to robustness.

Summary. In Sections 3.3 and 3.4, we will investigate stability of both natural data and adversarial examples by considering sampling from Gaussian distributions centered at a data point with varying standard deviations. Using the standard deviation as a parameter, we are able to derive a statistic for each point that captures how entrenched it is in its class in a way that is less restrictive than the robustness described by Khoury and Hadfield-Menell (2018), takes into account the rareness of adversarial

examples described by Hu et al. (2019), builds on the idea of sampling described by Roth, Kilcher, and Hofmann (2019) and Hosseini, Kannan, and Poovendran (2019), and represent curvatures in a sense related to Fawzi et al. (2018). Furthermore, we will relate these stability studies to direct measurement of interpolation incident angles with decision boundaries in Subsection 3.3.2 and 3.4.3 and the effect of reduction of data onto a known lower dimensional manifold in Subsections 3.4.5 and 3.4.4.

3.3 Methods

In this section we will lay out the theoretical framework for studying stability, persistence, and decision boundary corssing-angles.

3.3.1 Stability and Persistence

In this section we define a notion of stability of classification of a point under a given classification model. In the following, X represents the ambient space the data is drawn from (typically $\mathbb{R}^n$) even if the data lives on a submanifold of X, and L is a set of labels (often $\{1, \dots, \ell\}$). Note that points $x \in X$ can be natural or adversarial points.

Definition 3.3.1. *Let $\mathcal{C} : X \to L$ be a classifier, $x \in X$, $\gamma \in (0, 1)$, and $\sigma > 0$. We say x is* (γ, σ)-stable *with respect to $\mathcal{C}$ if $\mathbb{P}[\mathcal{C}(x') = \mathcal{C}(x)] \geq \gamma$ for $x' \sim \rho = N(x, \sigma^2 I)$; i.e. x' is drawn from a Gaussian with variance σ^2 and mean x.*

In the common setting when $X = \mathbb{R}^n$, we have

$$\mathbb{P}[\mathcal{C}(x') = \mathcal{C}(x)] = \int_{\mathbb{R}^n} \mathbb{1}_{\mathcal{C}^{-1}(\mathcal{C}(x))}(x') d\rho(x') = \rho(\mathcal{C}^{-1}\mathcal{C}(x)).$$

Note here that $\mathcal{C}^{-1}$ denotes preimage. One could substitute various probability measures ρ above with mean x and variance σ^2 to obtain different measures of stability corresponding to different ways of sampling the neighborhood of a point. Another natural choice would be sampling the uniform measure on balls of changing radius. Based on the concentration of measure for both of these families of measures we do not anticipate significant qualitative differences in these two approaches. We propose Gaussian sampling because it is also a product measure, which makes it easier to sample and simplifies some other calculations below.

For the Gaussian measure, the probability above may be written more concretely as

$$\frac{1}{\left(\sqrt{2\pi}\sigma\right)^n}\int_{\mathbb{R}^n}\mathbb{1}_{\mathcal{C}^{-1}(\mathcal{C}(x))}(x')e^{-\frac{\left|x-x'\right|^2}{2\sigma^2}}dx'. \tag{3.1}$$

In this work, we will conduct experiments in which we estimate this stability for fixed (γ, σ) pairs via a Monte Carlo sampling, in which case the integral (3.1) is approximated by taking N i.i.d. samples $x_k \sim \rho$ and computing

$$\frac{|x_k : \mathcal{C}(x_k) = \mathcal{C}(x)|}{N}.$$

Note that this quantity converges to the integral (3.1) as $N \to \infty$ by the Law of Large Numbers.

The ability to adjust the quantity γ is important because it is much weaker than a notion of stability that requires a ball that stays away from the decision boundary as by Khoury and Hadfield-Menell (2018). By choosing γ closer to 1, we can require the samples to be more within the same class, and by adjusting γ to be smaller we can allow more overlap.

We also propose a related statistic, *persistence*, by fixing a particular γ and adjusting σ. For any $x \in X$ not on the decision boundary, for any choice of $0 < \gamma < 1$ there exists a σ_γ small enough such that if $\sigma < \sigma_\gamma$ then x is (γ, σ)-stable. We can now take the largest such σ_γ to define persistence.

Definition 3.3.2. *Let $\mathcal{C} : X \to L$ be a classifier, $x \in X$, and $\gamma \in (0, 1)$. Let σ_γ^* be the maximum σ_γ such that x is (γ, σ)-stable with respect to $\mathcal{C}$ for all $\sigma < \sigma_\gamma$. We say that x has* γ-persistence σ_γ^*.

The γ-persistence quantity σ_γ^* measures the stability of the neighborhood of a given x with respect to the output classification. Small persistence indicates that the classifier is unstable in a small neighborhood of x, whereas large persistence indicates stability of the classifier in a small neighborhood of x. In the later experiments, we have generally taken $\gamma = 0.7$. This choice is arbitrary and chosen to fit the problems considered here. In our experiments, we did not see significant change in results with small changes in the choice of γ.

In our experiments, we numerically estimate γ-persistence via a bisection algorithm that we term the Bracketing Algorithm. Briefly, the algorithm first chooses search space bounds $\sigma_{\min}$ and $\sigma_{\max}$ such that x is $(\gamma, \sigma_{\min})$-stable but is not $(\gamma, \sigma_{\max})$-stable with respect to $\mathcal{C}$, and then proceeds to evaluate stability by bisection until an approximation of σ_γ^* is obtained.

3.3.2 Decision Boundaries

In order to examine decision boundaries and their properties, we will carefully define the decision boundary in a variety of equivalent formulations.

The Argmax Function Arises in Multi-Class Problems

A central issue when writing classifiers is mapping from continuous outputs or probabilities to discrete sets of classes. Frequently argmax type functions are used to accomplish this mapping. To discuss decision boundaries, we must precisely define argmax and some of its properties.

In practice, argmax is not strictly a function, but rather a mapping from the set of outputs or activations from another model into the power set of a discrete set of classes:

$$\text{argmax} : \mathbb{R}^k \to \mathcal{P}(L) \tag{3.2}$$

Defined this way, we cannot necessarily consider argmax to be a function in general as the singleton outputs of argmax overlap in an undefined way with other sets from the power set. However, if we restrict our domain carefully, we can identify certain properties. Restricting to only the pre-image of the singletons, it should be clear that argmax is constant. Indeed, restricted to the pre-image of any set in the power-set, argmax is constant and thus continuous. This induces the discrete topology whereby the pre-image of an individual singleton is open. Observe that for any point whose image is a singleton, one element of the domain vector must exceed the others by $\varepsilon > 0$. We shall use the ℓ^1 metric for distance, and thus if we restrict ourselves to a ball of radius ε, then all elements inside this ball will have that element still larger than the rest and thus map to the same singleton under argmax. Since the union of infinitely many open sets is open in $\mathbb{R}^k$, the union of all singleton pre-images is an open set. Conveniently this also provides proof that the union of all of the non-singleton sets in

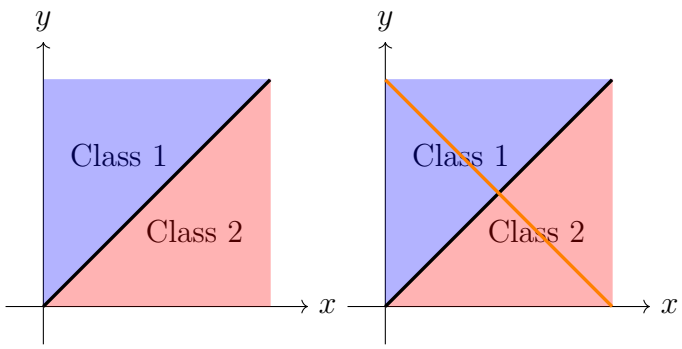


FIGURE 3.1: Decision boundary in $[0,1] \times [0,1]$ (left) and decision boundary restricted to probabilities (right) If the output of F are *probabilities* which add to one, then all points of x will map to the orange line on the right side of Figure 3.1. We note that the point $(0.5, 0.5)$ is therefore the only point on the decision boundary for probability valued F. We may generalize to higher dimensions where all probability valued models F will map into the the plane $x + y + z + \cdots = 1$ in Y and the decision boundary will be partitioned into $K - 1$ components, where the K-decision boundary is the intersection of this plane with the *centroid* line $x = y = z = \cdots$ and the 2-decision boundaries become planes intersecting at the same line.

$\mathcal{P}(C)$ is a closed set. We will call this closed set the argmax Decision Boundary. We will list two equivalent formulations for this boundary.

Complement Definition A point x is in the *decision interior* $D'_{\mathcal{C}}$ for a classifier $\mathcal{C} : \mathbb{R}^N - > L$ if there exists $\delta > 0$ such that $\forall \epsilon < \delta$, the number of elements $n(\mathcal{C}(B_\epsilon(x))) = 1$.

The *decision boundary* of a classifier $\mathcal{C}$ is the closure of the complement of the decision interior $\overline{\{x : x \notin D'_{\mathcal{C}}\}}$.

Level Set Definition For an input space X, the decision boundary $D \subset X$ of a probability valued function f is the pre-image of a union of all level sets of outputs

$f(X) = c_1, c_2, ..., c_k$ defined by a constant c such that for some set of indices I, we have $c = c_i$ for every i in I and $c > c_j$ for every j not in I. The pre-image of each such set are all x such that $f(x) = A_c$ for some c.

3.4 Experiments

In this section we investigate the stability and persistence behavior of natural and adversarial examples for MNIST (LeCun and Cortes, 2010) and ImageNet (Russakovsky et al., 2015) using a variety of different classifiers. For each set of image samples generated for a particular dataset, model, and attack protocol, we study (γ, σ)-stability and γ-persistence of both natural and adversarial images, and also compute persistence along trajectories from natural to adversarial images. In general, we use $\gamma = 0.7$, and note that the observed behavior does not change significantly for small changes in γ. While most of the adversarial attacks considered here have a clear target class, the measurement of persistence does not require considering a particular candidate class. Furthermore, we will evaluate decision boundary incidence angles and apply our conclusions to evaluate models trained with manifold aligned gradients.

3.4.1 MNIST Experiments

Since MNIST is relatively small compared to ImageNet, we trained several classifiers with various architectures and complexities and implemented the adversarial attacks directly. Adversarial examples were generated against each of these models using Iterative Gradient Sign Method (IGSM (Kurakin, Goodfellow, and Bengio, 2016))

and Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS (Liu and Nocedal, 1989)).

Investigation of (γ, σ)-stability on MNIST

We begin with a fully connected ReLU network with layers of size 784, 100, 20, and 10 and small regularization $\lambda = 10^{-7}$ which is trained on the standard MNIST training set. We then start with a randomly selected MNIST test image x_1 from the `1`'s class and generate adversarial examples $x_0, x_2, \ldots, x_9$ using IGSM for each target class other than `1`. The neighborhoods around each x_i are examined by generating 1000 i.i.d. samples from $N(x_i, \sigma^2 I)$ for each of 100 equally spaced standard deviations $\sigma \in (0, 1.6)$. Figure 3.2 shows the results of the Gaussian perturbations of a natural example x_1 of the class labeled `1` and the results of Gaussian perturbations of the adversarial example x_0 targeted at the class labeled `0`. We provide other examples of $x_2, \ldots, x_9$ in the supplementary materials. Note that the original image is very stable under perturbation, while the adversarial image is not.

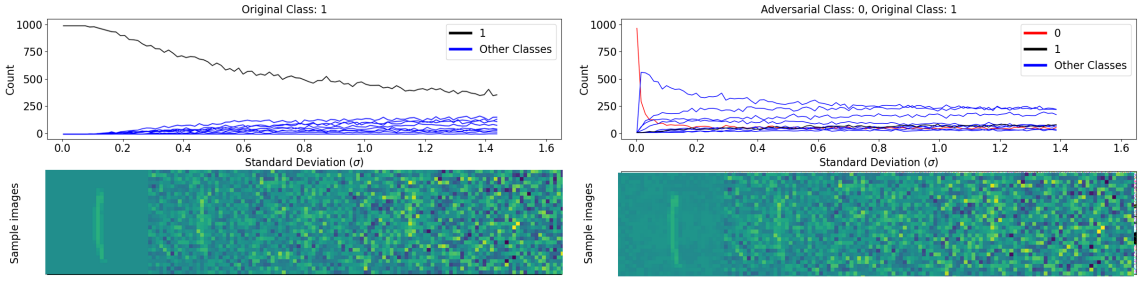


FIGURE 3.2: Frequency of each class in Gaussian samples with increasing variance around a natural image of class `1` (left) and around an adversarial attack of that image targeted at `0` generated using IGSM (right). The adversarial class (`0`) is shown as a red curve. The natural image class (`1`) is shown in black. Bottoms show example sample images at different standard deviations for natural (left) and adversarial (right) examples.

Persistence of adversarial examples for MNIST

To study persistence of adversarial examples on MNIST, we take the same network architecture as in the previous subsection and randomly select 200 MNIST images. For each image, we used IGSM to generate 9 adversarial examples (one for each target class) yielding a total of 1800 adversarial examples. In addition, we randomly sampled 1800 natural MNIST images. For each of the 3600 images, we computed 0.7-persistence; the results are shown in Figure 3.3. One sees that 0.7-persistence of adversarial examples tends to be significantly smaller than that of natural examples for this classifier, indicating that they are generally less stable than natural images. We will see subsequently that this behavior is typical.

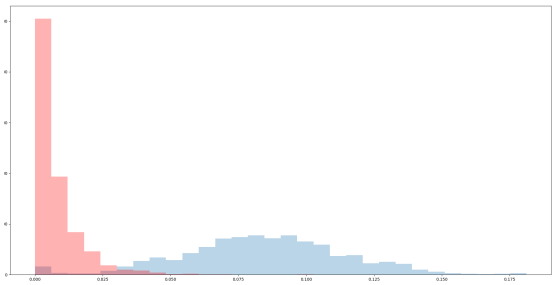


FIGURE 3.3: Histogram of 0.7-persistence of IGSM-based adversarial examples (red) and natural examples (blue) on MNIST.

Next, we investigate the relationship of network complexity and (γ, σ)-stability by revisiting the now classic work of Szegedy et al. (2014) on adversarial examples.

Table 3.1 recreates and adds on to part of Szegedy et al., 2014, Table 1 in which networks of differing complexity are trained and attacked using L-BFGS. The table contains new columns showing the average 0.7-persistence for both natural and adversarial examples for each network, as well as the average distortion for the adversarial examples. The distortion is the ℓ^2-norm divided by square root of the

dimension n. The first networks listed are of the form FC10-k, and are fully connected single layer ReLU networks that map each input vector $x \in \mathbb{R}^{784}$ to an output vector $y \in \mathbb{R}^{10}$ with a regularization added to the objective function of the form $\lambda \|w\|_2 / N$, where $\lambda = 10^{-k}$ and N is the number of parameters in the weight vector w defining the network. The higher values of λ indicate more regularization.

FC100-100-10 and FC200-200-10 are networks with 2 hidden layers (with 100 and 200 nodes, respectively) with regularization added for each layer of perceptrons with the λ for each layer equal to $10^{-5}, 10^{-5}$, and 10^{-6}. Training for these networks was conducted with a fixed number of epochs (typically 21). For the bottom half of Table 3.1, we also considered networks with four convolutional layers plus a max-pooling layer connected by ReLU to a fully connected hidden layer with increasing numbers of channels denoted as as "C-Ch," where C reflects that this is a CNN and Ch denotes the number of channels. A more detailed description of these networks can be found in Appendix B.3.

The main observation from Table 3.1 is that for higher complexity networks, adversarial examples tend to have smaller persistence than natural examples. Histograms reflecting these observations can be found in the supplemental material. Another notable takeaway is that for models with fewer effective parameters, the attack distortion necessary to generate a successful attack is so great that the resulting image is often more stable than a natural image under that model, as seen particularly in the FC10 networks. Once there are sufficiently many parameters available in the neural network, we found that both the average distortion of the adversarial examples and the average 0.7-persistence of the adversarial examples tended to be smaller. This observation is consistent with the idea that networks with more parameters are more

TABLE 3.1: Recreation of Szegedy et al. (2014, Table 1) for the MNIST dataset. For each network, we show Testing Accuracy (in %), Average Distortion ($\|x\|_2/\sqrt{n}$) of adversarial examples, and new columns show average 0.7-persistence values for natural (Nat) and adversarial (Adv) images. 300 natural and 300 adversarial examples generated with L-BFGS were used for each aggregation.

Network	Test Acc	Avg Dist	Persist (Nat)	Persist (Adv)
FC10-4	92.09	0.123	0.93	1.68
FC10-2	90.77	0.178	1.37	4.25
FC10-0	86.89	0.278	1.92	12.22
FC100-100-10	97.31	0.086	0.65	0.56
FC200-200-10	97.61	0.087	0.73	0.56
C-2	95.94	0.09	3.33	0.027
C-4	97.36	0.12	0.35	0.027
C-8	98.50	0.11	0.43	0.0517
C-16	98.90	0.11	0.53	0.0994
C-32	98.96	0.11	0.78	0.0836
C-64	99.00	0.10	0.81	0.0865
C-128	99.17	0.11	0.77	0.0883
C-256	99.09	0.11	0.83	0.0900
C-512	99.22	0.10	0.793	0.0929

likely to exhibit decision boundaries with more curvature.

3.4.2 Results on ImageNet

For ImageNet (Deng et al., 2009), we used pre-trained ImageNet classification models, including alexnet (Krizhevsky, Sutskever, and Hinton, 2012) and vgg16 (Simonyan and Zisserman, 2014).

We then generated attacks based on the ILSVRC 2015 (Russakovsky et al., 2015) validation images for each of these networks using a variety of modern attack protocols, including Fast Gradient Sign Method (FGSM (Goodfellow, Shlens, and Szegedy,

2014)), Momentum Iterative FGSM (MIFGSM ("Boosting Adversarial Attacks With Momentum")), Basic Iterative Method (BIM (Kurakin, Goodfellow, and Bengio, 2016)), Projected Gradient Descent (PGD (Madry et al., 2017)), Randomized FGSM (R+FGSM ("Ensemble Adversarial Training: Attacks and Defenses")), and Carlini-Wagner (CW Carlini and Wagner (2016)). These were all generated using the TorchAttacks by Kim (2020) toolset.

Investigation of (γ, σ)-stability on ImageNet

In this section, we show the results of Gaussian neighborhood sampling in ImageNet. Figures 3.4 and 3.5 arise from vgg16 and adversarial examples created with BIM; results for other networks and attack strategies are similar, with additional figures in the supplementary material. Figure 3.4 (left) begins with an image x with label `goldfinch`. For each equally spaced $\sigma \in (0, 2)$, 100 i.i.d. samples were drawn from the Gaussian distribution $N(x, \sigma^2 I)$, and the counts of the vgg16 classification for each label are shown. In Figure 3.4 (right), we see the same plot, but for an adversarial example targeted at the class `indigo_bunting`, which is another type of bird, using the BIM attack protocol.

The key observation in Figure 3.4 is that the frequency of the class of the adversarial example (`indigo_bunting`, shown in red) falls off much quicker than the class for the natural example (`goldfinch`, shown in black). In this particular example, the original class appears again after the adversarial class becomes less prevalent, but only for a short period of σ, after which other classes begin to dominate. In some examples the original class does not dominate at all after the decline of the adversarial class. The adversarial class almost never dominates for a long period of σ.

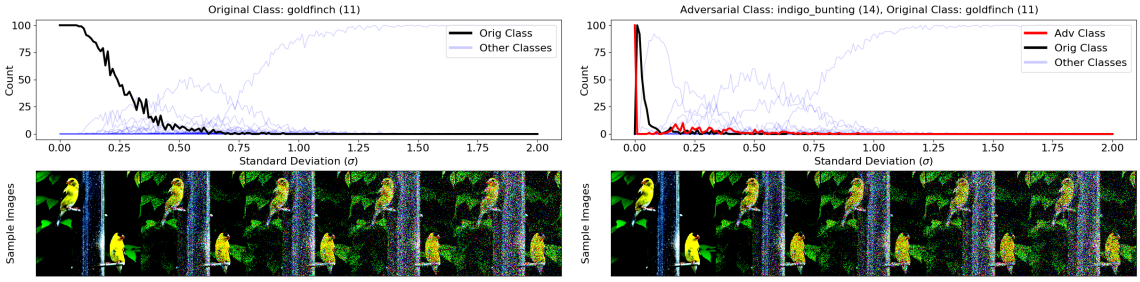


FIGURE 3.4: Frequency of each class in Gaussian samples with increasing variance around a `goldfinch` image (left) and an adversarial example of that image targeted at the `indigo_bunting` class and calculated using the BIM attack (right). Bottoms show example sample images at different standard deviations for natural (left) and adversarial (right) examples.

Persistence of adversarial examples on ImageNet

Figure 3.5 shows a plot of the 0.7-persistence along the straight-line path between a natural example and adversarial example as parametrized between 0 and 1. It can be seen that the dropoff of persistence occurs precisely around the decision boundary. This indicates some sort of curvature favoring the class of the natural example, since otherwise the persistence would be roughly the same as the decision boundary is crossed.

An aggregation of persistence for many randomly selected images from the `goldfinch` class in the validation set for Imagenet are presented in Table 3.2. For each image of a `goldfinch` and for each network of alexnet and vgg16, attacks were prepared to a variety of 28 randomly selected targets using a BIM, MIFGSM, PGD, FGSM, R+FGSM, and CW attack strategies. The successful attacks were aggregated and their 0.7-persistences were computed using the Bracketing Algorithm along with the 0.7-persistences of the original images from which each attack was generated. Each attack strategy had a slightly different mixture of which source image and attack

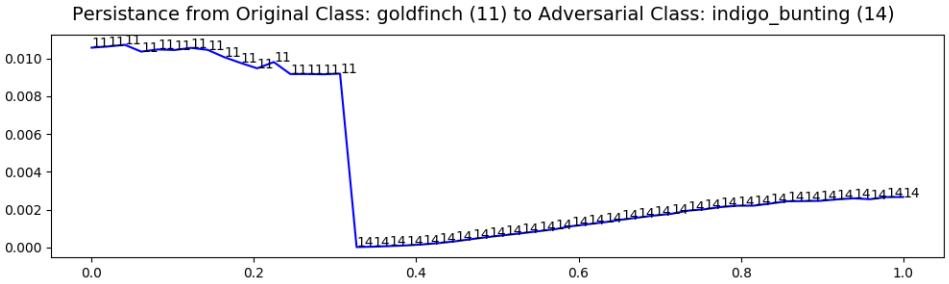


FIGURE 3.5: The 0.7-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) on a vgg16 classifier. The classification of each image on the straight line is listed as a number so that it is possible to see the transition from one class to another. The vertical axis is 0.7-persistence and the horizontal axis is progress towards the adversarial image.

Network/Method	Avg Dist	Persist (Nat)	Persist (Adv)
alexnet (total)	0.0194	0.0155	0.0049
BIM	0.0188	0.0162	0.0050
MIFGSM	0.0240	0.0159	0.0053
PGD	0.0188	0.0162	0.0050
vgg16 (total)	0.0154	0.0146	0.0011
BIM	0.0181	0.0145	0.0012
MIFGSM	0.0238	0.0149	0.0018
PGD	0.0181	0.0145	0.0012

TABLE 3.2: The 0.7-persistence values for natural (Nat) and adversarial (Adv) images along with average distortion for adversarial images of alexnet and vgg16 for attacks generated with BIM, MIFGSM, and PGD on images from class `goldfinch` targeted toward other classes from the ILSVRC 2015 classification labels.

target combinations resulted in successful attacks. The overall rates for each are listed, as well as particular results on the most successful attack strategies in our experiments, BIM, MIFGSM, and PGD. The results indicate that adversarial images generated for these networks (alexnet and vgg16) using these attacks were less persistent, and hence

less stable, than natural images for the same models.

3.4.3 Decision Boundary Interpolation and Angle Measurement

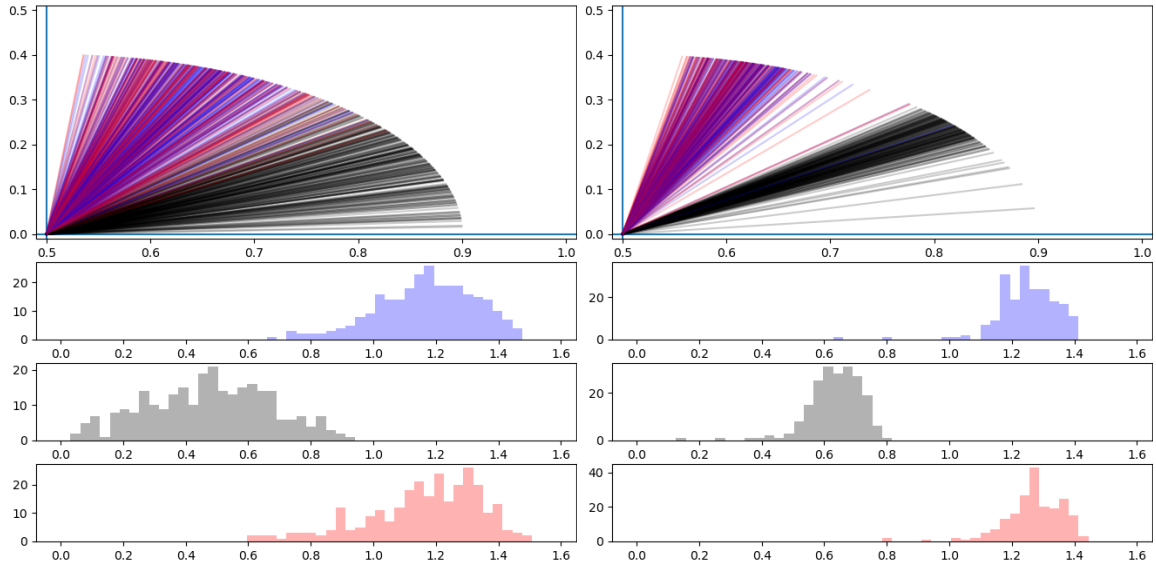


FIGURE 3.6: Decision boundary incident angles between test to test interpolation and a computed normal vector to the decision boundary images (left) and between test and adversarial images (right). Angles (plotted Top) are referenced to decision boundary so $\pi/2$ radians (right limit of plots) corresponds with perfect orthogonality to decision boundary. Lines and histograms measure angles of training gradients (Top) linear interpolant (Middle) and adversarial gradients (Bottom). x and y axes are the axes of the unit-circle so angles can be compared. All angles are plotted in the upper-right quadrant for brevity. The lower plots are all histograms with their y axes noting counts and their x-axes showing angles all projected to the range from 0 to $\pi/2$.

In order to understand this sudden drop in persistence across the decision boundary observed in Figure 3.5, we will investigate incident angle of the interpolation with the decision boundary. In order to measure these angles, we must first interpolate along the decision boundary between two points. We will do this for pairs of test

and test and pairs of test and adversary. In both cases, we will use a bracketing algorithm along the interpolation from candidate points to identify a point within machine-precision of the decision boundary x_b.

Next, we will take 5000 samples from a Gaussian centered at this point with small standard deviation $\sigma = 10^{-6}$. Next, for each sample, we will perform an adversarial attack in order to produce a corresponding point on the opposite side of the decision boundary. Now for this new pair (sample and attacked sample), we will repeat the interpolation bracketing procedure in order to obtain the projection of this sample onto the decision boundary along the attack trajectory. Next, we will use singular value decomposition (SVD) on the differences between the projected samples and our decision boundary point x_b to compute singular values and vectors from these projected samples. We will use the right singular vector corresponding with the smallest singular value as an approximation of a normal vector to the decision boundary at x_b. This point is difficult to compute due to degeneracy of SVD for small singular values, however in our tests, this value could be computed to a precision of 0.003. We will see that this level of precision exceeds that needed for the angles computed with respect to this normal vector sufficiently.

From Figure 3.6 we notice that neither training gradients nor adversarial gradients are orthogonal to the decision boundary. From a theory perspective, this is possible because this problem has more than 2 classes, so that the decision boundary includes $(0.34, 0.34, 0.32)$ and $(0.4, 0.4, 0.2)$. That is to say that the level set definition of the decision boundary has degrees of freedom that do not require orthogonality of gradients. More interestingly, both natural and adversarial linear interpolants tend to cross at acute angles with respect to the decision boundary, with adversarial attacks tending to

be closer to orthogonal. This suggests that obliqueness of the decision boundary with respect to test points may be related to adversarial vulnerability. We will leverage this understanding with manifold alignment to see if constraining gradients to a lower dimensional manifold, and thus increasing orthogonality of gradients will increase robustness.

3.4.4 Manifold Alignment on MNIST via PCA

In order to provide an empirical measure of alignment, we first require a well defined image manifold. The task of discovering the true structure of k-dimensional manifolds in $\mathbb{R}^d$ given a set of points sampled on the manifold has been studied previously (Khoury and Hadfield-Menell, 2018). Many algorithms produce solutions which are provably accurate under data density constraints. Unfortunately, these algorithms have difficulty extending to domains with large d due to the curse of dimensionality. Our solution to this fundamental problem is to sidestep it entirely by redefining our dataset. We begin by projecting our data onto a well known low dimensional manifold, which we can then measure with certainty.

We first fit a PCA model on all training data, using k components for each class to form a component matrix W, where $k << d$. Given the original dataset X, we create a new dataset $X_{\mathcal{M}} := \{x \times \mathbf{W}^T \times \mathbf{W} : x \in X\}$. We will refer to this set of component vectors as $\mathbf{W}$. Because the rank of the linear transformation matrix, k, is defined lower than the dimension of the input space, d, this creates a dataset which lies on a linear subspace of $\mathbb{R}^d$. This subspace is defined by the span of $X \times \mathbf{W}^T$ and any vector in $\mathbb{R}^d$ can be projected onto it. Any data point drawn from $\{z \times \mathbf{W}^T : z \in \mathbb{R}^k\}$

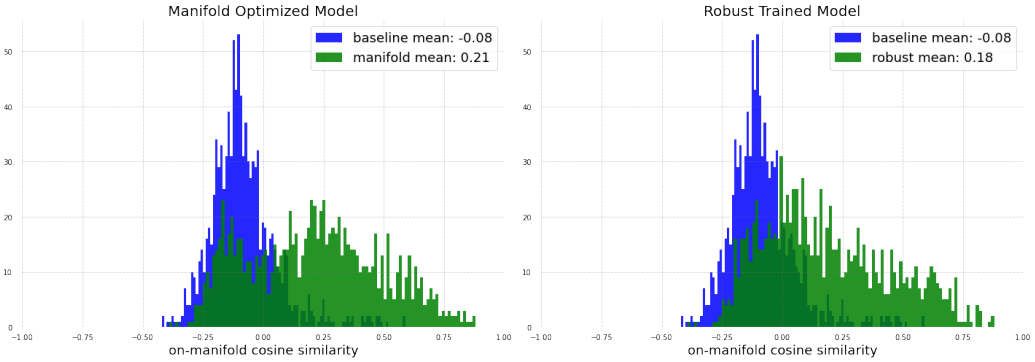


FIGURE 3.7: Comparison of on-manifold components between baseline network, robust trained models, and manifold optimized models. Large values indicate higher similarity to the manifold. Y-axes for both plots are histogram counts. Both robust and manifold optimized models are more 'on-manifold' than the baseline, with adversarial training being slightly less so.

is considered a valid datapoint. This gives us a continuous linear subspace which can be used as a data manifold.

Given that it our goal to study the simplest possible case, we chose MNIST as the dataset to be projected and selected $k = 28$ components. We refer to this new dataset as Projected MNIST (PMNIST). The true rank of PMNIST is lower than that of the original MNIST data, meaning there was information lost in this projection. The remaining information we found is sufficient to achieve 92% accuracy using a baseline Multylayer Perceptron (MLP), and the resulting images retain their semantic properties as shown in Figure 3.9.

3.4.5 Manifold Aligned Gradients

Component vectors extracted from the original dataset are used to project gradient examples onto our pre-defined image manifold.

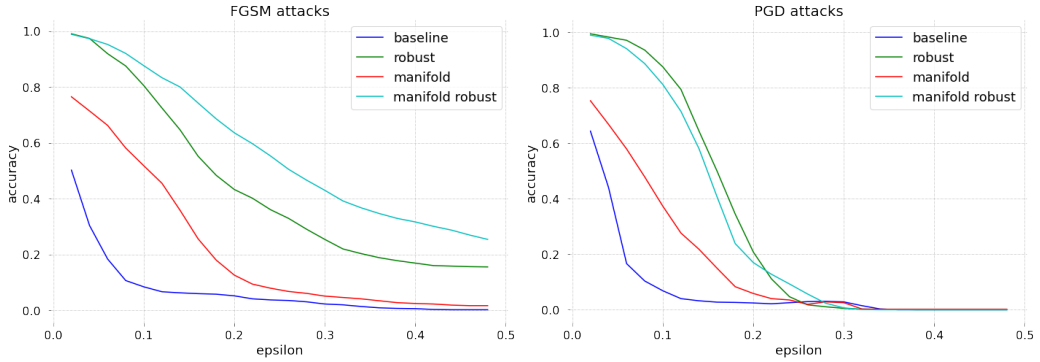


FIGURE 3.8: Comparison of adversarial robustness for PMNIST models under various training conditions. Attacks are prepared using a range of a distortion parameter epsilon. For FGSM, the sign of the gradient is multiplied by each epsilon. For PGD, epsilon is determined by a weight on the l_2 norm term of the adversarial loss function. Many variations of the l_2 weight are performed, and then they are aggregated and the distance of each perturbation is plotted as epsilon. For both FGSM and PGD, we see a slight increase in robustness from using manifold optimization. Adversarial training still improves performance significantly more than manifold optimization. Another observation to note is that when both the manifold, and adversarial objective were optimized, increased robustness against FGSM attacks was observed. All robust models were trained using the l_∞ norm at epsilon = 0.1.

Given a gradient example $\nabla_x = \frac{\partial f_\theta(x,y)}{\partial x}$ where f_θ represents a neural network parameterized by weights θ, ∇_x is transformed using the coefficient vectors $\mathbf{W}$.

$$\rho_x = \nabla_x \times \mathbf{W}^T \times \mathbf{W} \tag{3.3}$$

The projection of the original vector onto this new transformed vector will be referred to as $P_\mathcal{M}$. The ratio of norms of this projection gives a metric of manifold alignment:

$$\frac{||\nabla_x||}{||P_\mathcal{M}(\nabla_x)||}. \tag{3.4}$$

This gives us a way of measuring the ratio between on-manifold and off-manifold

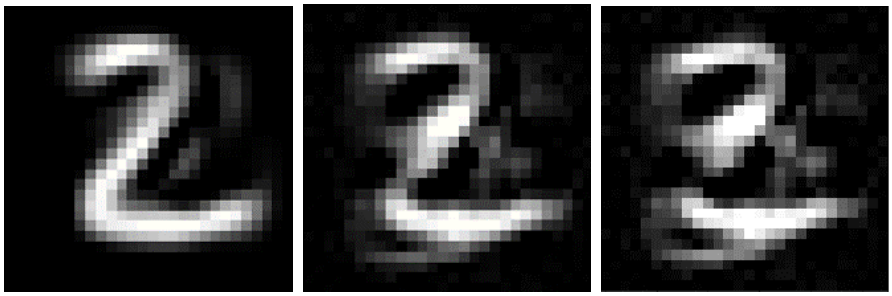


FIGURE 3.9: Visual example of manifold optimized model transforming 2 into 3. Original PMNIST image on left, center image is center point between original and attacked, on right is the attacked image. Transformation performed using PGD using the l_∞ norm. Visual evidence of manifold alignment is often subjective and difficult to quantify. This example is provided as a baseline to substantiate our claim that our empirical measurements of alignment are valid.

components of the gradient. Additionally, both cosine similarity and the vector rejection were also tested but the norm ratio we found to be the most stable in training. We use this measure as both a metric and a loss, allowing us to optimize the following objective:

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[L(\theta,x,y)+\alpha\frac{||\nabla_x||}{||P_{\mathcal{M}}(\nabla_x)||}\right] \tag{3.5}$$

Where $L(\theta,x,y)$ represents our classification loss term and α is a hyper parameter determining the weight of the manifold loss term.

3.4.6 Manifold Alignment Robustness Results

All models were two layer MLPs with 1568 nodes in each hidden layer. The hidden layer size was chosen as twice the input size. This arrangement was chosen to maintain the simplest possible case.

Two types of attacks were leveraged in this study: fast gradient sign method (FGSM) (Goodfellow, Shlens, and Szegedy, 2014) which performs a single step based on the sign of an adversarial gradient for each input and projected gradient descent (PGD) which performs gradient descent on input data using adversarial gradients in order to produce adversarial attacks (Madry et al., 2017). A total of four models were trained and evaluated on these attacks: Baseline, Robust, Manifold and Manifold Robust. All models, including the baseline, were trained on PMNIST (a fixed permutation is applied to the training and test images of the MNIST dataset). “Robust" in our case refers to models trained with new adversarial examples labeled for their class *before* perturbation during each epoch consistent with Tramer and Boneh (2019). All robust models were trained using the l_∞ norm and a maximum perturbation parameter of $\epsilon = 0.1$. Manifold Robust refers to both optimizing our manifold objective and robust training simultaneously.

Figure 3.7 shows the cosine similarity of the gradient and its projection onto the reduced space W on the testing set of PMNIST for both the Manifold model and Robust model. Higher values (closer to 1) indicate the model is more aligned with the manifold. Both robust and MAG models here are shown to be more on manifold than the Baseline. This demonstrates that our metric for alignment is being optimized as a consequence of adversarial training.

Figure 3.8 shows the adversarial robustness of each model. In both cases, aligning the model to the manifold shows an increase in accuracy of classification for adversarial images over the baseline. However, we do not consider the performance boost against PGD to be significant enough to call these models robust against PGD attacks. Another point of interest that while using both our manifold alignment metric and

adversarial training, we see an even greater improvement against FGSM attacks.

The fact that this performance increase is not shared by PGD training may indicate a relationship between these methods. Our current hypothesis is that a linear representation of the image manifold is sufficient to defend against linear attacks such as FGSM, but cannot defend against a non-linear adversary.

3.5 Conclusion

In order to better understand the observed tendency for points near natural data to be classified similarly and points near adversarial examples to be classified differently, we defined a notion of (γ, σ)-stability which is easily estimated by Monte Carlo sampling. For any data point x, we then define the γ-persistence to to be the smallest σ_γ such that the probability of similarly classified data is at least γ when sampling from Gaussian distributions with mean x and standard deviation less than σ_γ. The persistence value can be quickly estimated by a Bracketing Algorithm. These two measures were considered with regard to both the MNIST and ImageNet datasets and with respect to a variety of classifiers and adversarial attacks. We found that adversarial examples were much less stable than natural examples in that the 0.7-persistence for natural data was usually significantly larger than the 0.7-persistence for adversarial examples. We also saw that the dropoff of the persistence tends to happen precisely near the decision boundary. Each of these observations is strong evidence toward the hypothesis that adversarial examples arise inside cones or high curvature regions in the adversarial class, whereas natural images lie outside such regions.

We also found that often the most likely class for perturbations of an adversarial examples is a class other than the class of the original natural example used to generate the adversarial example; instead, some other background class is favored. In addition, we found that some adversarial examples may be more stable than others, and a more detailed probing using the concept of (γ, σ)-stability and the γ-persistence statistic may be able to help with a more nuanced understanding of the geometry and curvature of the decision boundary. Although not pursued here, the observations and statistics used in this paper could potentially be used to develop methods to detect adversarial examples as in (Crecchi, Bacciu, and Biggio, 2019; Frosst, Sabour, and Hinton, 2018; Hosseini, Kannan, and Poovendran, 2019; Lee et al., 2018b; "Detecting and Diagnosing Adversarial Images with Class-Conditional Capsule Reconstructions"; Roth, Kilcher, and Hofmann, 2019) and others. As with other methods of detection, this may be susceptible to adaptive attacks as discussed by Tramèr et al. ("On Adaptive Attacks to Adversarial Example Defenses").

For the future, we have made several observations: We found that some adversarial examples may be more stable than others. More detailed probing using the concept of (γ, σ)-stability and the γ-persistence along linear interpolation between natural images and between natural and adversarial images reveals sharp drops in persistence. Sharp drops in persistence correspond with oblique angles of incidence between linear interpolation vectors and the decision boundary learned by neural networks. Combining these observations, we can form a conjecture: Adversarial examples appear to exist near regions surrounded by negatively curved structures bounded by decision surfaces with relatively small angles relative to linear interpolation among training and testing data. This conjecture compares with the dimpled manifold hypothesis

(Shamir, Melamed, and BenShmuel, 2021), however our techniques provide geometric information that allows us to gain a more detailed analysis of this region than in that work. In addition, our analysis of gradient alignment with manifolds reinforces the notion that the obliqueness we observe may be a property which can be isolated and trained out of neural networks to some extent. Future work should focus on refining this conjecture with further tools to complete the spatial and mathematical picture surrounding adversarial examples.

Chapter 4 An Exact Kernel Equivalence for Finite Classification Models

In the process of trying to understand adversarial examples in a geometric sense, a question arises: How can we directly extract geometric properties from machine learning models. In this line of thought, kernel methods are particularly appealing because they implicitly define a spatial transform which is used to make predictions. The kernel, a symmetric positive-definite bilinear map, at the core of all kernel methods can be written as an inner-product in an appropriate Hilbert space for all problems. This is a spatial metric! Furthermore, kernel methods make predictions by comparing a test point (using the kernel) with all known training points:

$$P(x) = b + \sum_i K(x, x_i) \tag{4.1}$$

Any prediction can be decomposed into the *spatial* contribution from each training point. This would be an incredible way to interpret machine learning predictions including adversarial examples. My gut told me that connections here would be both possible and useful, leading to my careful review of the Neural-Tangent-Kernel and related literature eventually leading to an unpublished arxiv paper by Domingos (2020a).

The resulting paper which is presented here exactly as accepted to the archival

proceedings track of the Topology, Algebra, and Geometry Workshop at the International Conference on Machine Learning (ICML) 2023 proposes the first exact path kernel representation for general gradient trained classifiers. The primary derivation and proof was written by Brian Bell and the supporting implementation and work were mostly conducted by Michael Geyer. The central focus of the paper is on the derivation and demonstration that this method works in practice. The interest that gave rise for this approach comes from the fact that kernel methods and more specifically bilinear map based models decompose their predictions into a contribution from each of their training data. the Neural Tangent Kernel (NTK) is an interesting tool, but predicated on too many approximation assumptions. This exact formulation allows a much more solid foundation for analyzing neural networks and suggests the possibility that predictions can be decomposed using this framework. The implementation and application of this method to real machine learning models and tasks demonstrates that it is not only a theoretical framework; it is practical! As stated above, this paper was accepted for archival publication at the Topology Algebra and Geometry (TAG) workshop at ICML 2023 in Honolulu, Hawaii.

4.1 Introduction

This study investigates the relationship between kernel methods and finite parametric models. To date, interpreting the predictions of complex models, like neural networks, has proven to be challenging. Prior work has shown that the inference-time predictions of a neural network can be exactly written as a sum of independent predictions computed with respect to each training point. We formally show that classification

models trained with cross-entropy loss can be exactly formulated as a kernel machine. It is our hope that these new theoretical results will open new research directions in the interpretation of neural network behavior.

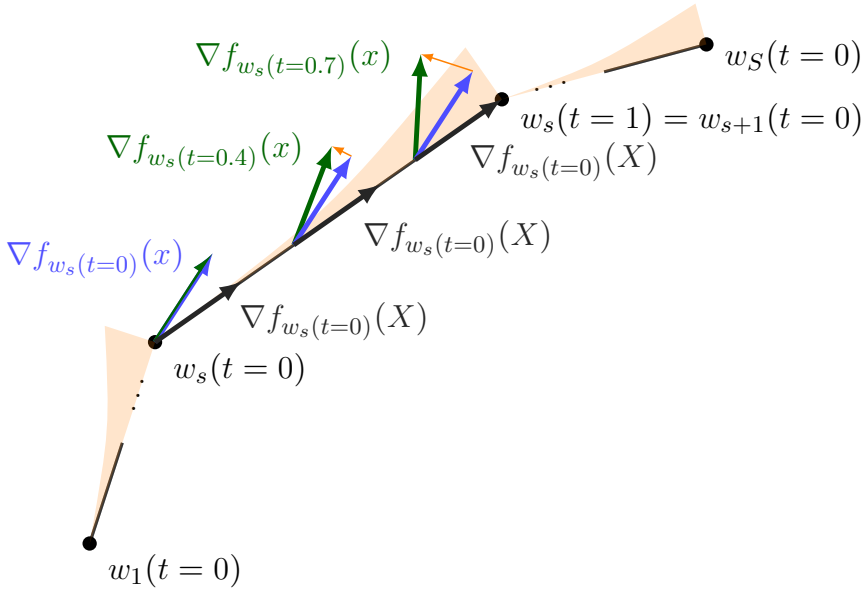


FIGURE 4.1: Comparison of test gradients used by Discrete Path Kernel (DPK) from prior work (Blue) and the Exact Path Kernel (EPK) proposed in this work (green) versus total training vectors (black) used for both kernel formulations along a discrete training path with S steps. Orange shading indicates cosine error of DPK test gradients versus EPK test gradients shown in practice in Fig. 4.2.

There has recently been a surge of interest in the connection between neural networks and kernel methods (Bietti and Mairal, 2019; Du et al., 2019; Tancik et al., 2020; Abdar et al., 2021; Geifman et al., 2020; Chen et al., 2020; Alemohammad et al., 2021). Much of this work has been motivated by the neural tangent kernel (NTK), which describes the training dynamics of neural networks in the infinite limit of network width (Jacot, Gabriel, and Hongler, 2018). We argue that many intriguing behaviors arise in the *finite* parameter regime which should, for example, satisfy

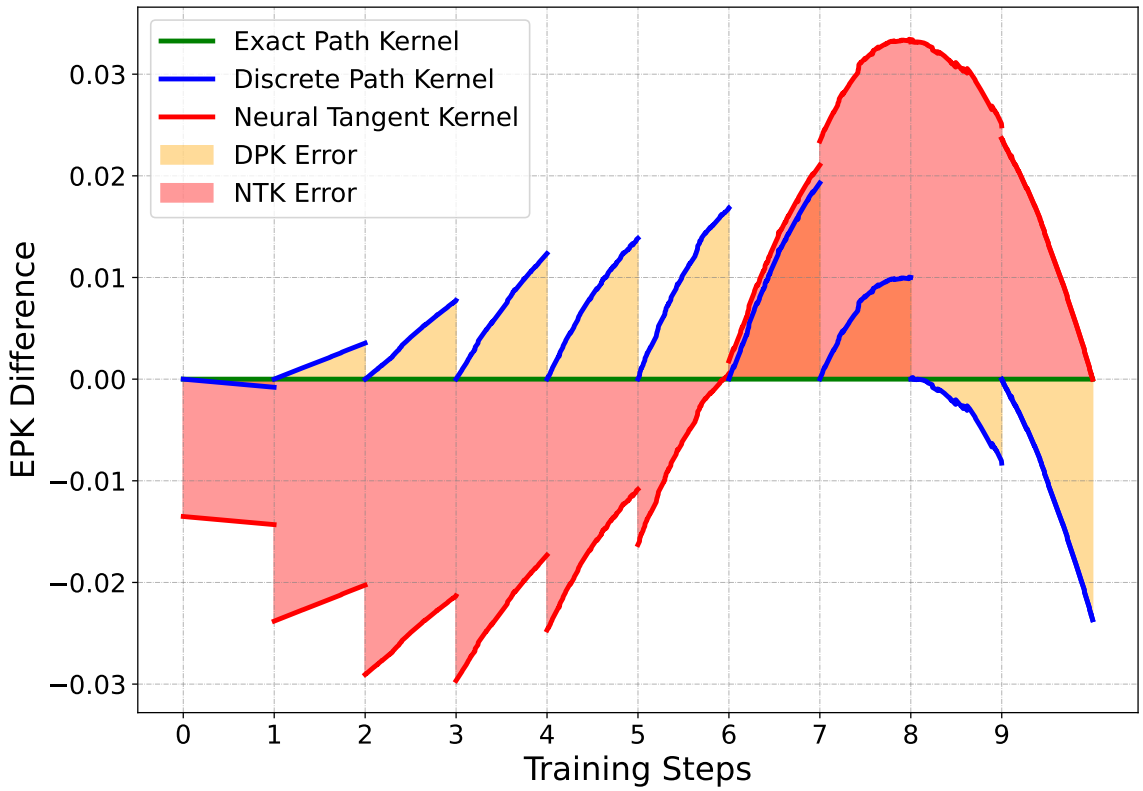


FIGURE 4.2: Measurement of gradient alignment on test points across the training path. The EPK is used as a frame of reference. The y-axis is exactly the difference between the EPK and other representations. For example $EPK - DPK = \langle \phi_{s,t}(X), \phi_{s,t}(x) - \phi_{s,0}(x) \rangle$ (See Definition 3.4). Shaded regions indicate total accumulated error. Note: this is measuring an angle of error in weight space; therefore, equivalent positive and negative error will not result in zero error.

the universal law of robustness proposed by Bubeck and Sellke (2021). All prior works, to the best of our knowledge, appeal to discrete approximations of the kernel corresponding to a neural network. Specifically, prior approaches are derived under the assumption that training step size is small enough to guarantee close approximation of a gradient flow (Ghojogh et al., 2021; Shawe-Taylor, Cristianini, et al., 2004; Zhao and Grishman, 2005).

In this work, we show that the simplifying assumptions used in prior works (i.e. infinite network width and infinitesimal gradient descent steps) are not necessary. Our **Exact Path Kernel (EPK)** provides the first exact method to study the behavior of finite-sized neural networks used for classification. Previous results are limited in application (Incudini et al., 2022) due to dependence of the kernel on test data unless strong conditions are imposed on the training process as by Chen et al. (2021b). We show, however, that the training step sizes used in practice do not closely follow this gradient flow, introducing significant error into all prior approaches (Figure 4.2).

Our experimental results build on prior studies attempting to evaluate empirical properties of the kernels corresponding to finite neural networks (Lee et al., 2018a; Chen et al., 2021b). While the properties of infinite neural networks are fairly well understood (Neal and Neal, 1996), we find that the kernels learned by finite neural networks have non-intuitive properties that may explain the failures of modern neural networks on important tasks such as robust classification and calibration on out-of-distribution data.

This paper makes the following significant theoretical and experimental contributions:

1. We prove that finite-sized neural networks trained with finite-sized gradient descent steps and cross-entropy loss can be exactly represented as kernel machines using the EPK. Our derivation incorporates a previously-proposed path kernel, but extends this method to account for practical training procedures (Domingos, 2020a; Chen et al., 2021b).

2. We demonstrate that it is computationally tractable to estimate the kernel underlying a neural network classifier, including for small convolutional computer vision models.

3. We compute Gram matrices using the EPK and use them to illuminate prior theory of neural networks and their understanding of uncertainty.

4. We employ Gaussian processes to compute the covariance of a neural network's logits and show that this reiterates previously observed shortcomings of neural network generalization.

4.2 Related Work

Fundamentally, the neural tangent kernel (NTK) is rooted in the concept that all information necessary to represent a parametric model is stored in the Hilbert space occupied by the model's weight gradients up to a constant factor. This is very well supported in infinite width (Jacot, Gabriel, and Hongler, 2018). In this setting, it has been shown that neural networks are equivalent to support vector machines, drawing a connection to maximum margin classifiers (Chen et al., 2021b; Chizat and Bach, 2020). Shah, Jain, and Netrapalli (2021) demonstrate that this maximum margin classifier exists in Wasserstien space; however, they also show that model gradients may not contain the required information to represent this.

The correspondence between kernel machines and parametric models trained by gradient descent has been previously developed in the case of a continuous training path (i.e. the limit as gradient descent step size $\varepsilon \to 0$) (Domingos, 2020b). We will refer to the previous finite approximation of this kernel as the Discrete Path Kernel (DPK). However, a limitation of this formulation is its reliance on a continuous integration over a gradient flow, which differs from the discrete forward Euler steps employed in real-world model training. This discrepancy raises concerns regarding the

applicability of the continuous path kernel to practical scenarios (Incudini et al., 2022). Moreover, the formulation of the sample weights and bias term in the DPK depends on its test points. Chen et al. (2021b) propose that this can be addressed, in part, by imposing restrictions on the loss function used for training, but did not entirely disentangle the kernel formulation from sample importance weights on training points (Chen et al., 2021b).

We address the limitations of Domingos (2020b) and Chen et al. (2021b) in Subsection 4.3.5. By default, their approach produces a system which can be viewed as an ensemble of kernel machines, but without a single aggregated kernel which can be analyzed directly. Chen et al. (2021b) propose that the resulting sum over kernel machines can be formulated as a kernel machine so long as the sign of the gradient of the loss stays constant through training; however, we show that this is not necessarily a sufficient restriction. Instead, their formulation leads to one of several non-symmetric functions which can serve as a surrogate to replicate a given models behavior, but without retaining properties of a kernel.

4.3 Theoretical Results

Our goal is to show an equivalence between any given finite parametric model trained with gradient descent $f_w(x)$ (e.g. neural networks) and a kernel based prediction that we construct. We define this equivalence in terms of the output of the parametric model $f_w(x)$ and our kernel method in the sense that they form identical maps from input to output. In the specific case of neural network classification models, we consider the mapping $f_w(x)$ to include all layers of the neural network up to and

including the log-softmax activation function. Formally:

Definition 4.3.1. *A kernel is a function of two variables which is symmetric and positive semi-definite.*

Definition 4.3.2. *Given a Hilbert space X, a test point $x \in X$, and a training set $X_T = \{x_1, x_2, ..., x_n\} \subset X$ indexed by I, a* Kernel Machine *is a model characterized by*

$$K(x) = b + \sum_{i=1}^{n} a_i k(x, x_i) \tag{4.2}$$

where the $a_i \in \mathbb{R}$ do not depend on x, $b \in \mathbb{R}$ is a constant, and k is a kernel (Rasmussen, Williams, et al., 2006).

By Mercer's Theorem (Ghojogh et al., 2021) a kernel can be produced by composing an inner product on a Hilbert space with a mapping ϕ from the space of data into the chosen Hilbert space. We use this property to construct a kernel machine of the following form.

$$\mathrm{K}(x) = b + \sum_{i \in I} a_i \langle \phi(x), \phi(x_i) \rangle \tag{4.3}$$

Where ϕ is a function mapping input data into the weight space via gradients. Our ϕ will additionally differentiate between test and training points to resolve a discontinuity that arises under discrete training.

4.3.1 Exact Path Kernels

We first derive a kernel which is an exact representation of the change in model output over one training step, and then compose our final representation by summing along the finitely many steps. Models trained by gradient descent can be characterized by a discrete set of intermediate states in the space of their parameters. These discrete states are often considered to be an estimation of the gradient flow, however in practical settings where $\epsilon > 0$ these discrete states differ from the true gradient flow. Our primary theoretical contribution is an algorithm which accounts for this difference by observing the true path the model followed during training. Here we consider the training dynamics of practical gradient descent steps by integrating a discrete path for weights whose states differ from the gradient flow induced by the training set.

Gradient Along Training Path vs Gradient Field: In order to compute the EPK, gradients on training data must serve two purposes. First, they are the reference points for comparison (via inner product) with test points. Second, they determine the path of the model in weight space. In practice, the path followed during gradient descent does not match the gradient field exactly. Instead, the gradient used to move the state of the model forward during training is only computed for finitely many discrete weight states of the model. In order to produce a path kernel, we must *continuously* compare the model's gradient at test points with *fixed* training gradients along each discrete training step s whose weights we we interpolate linearly by $w_s(t) = w_s - t(w_s - w_{s+1})$. We will do this by integrating across the gradient field induced by test points, but holding each training gradient fixed along the entire discrete step taken. This creates an asymmetry, where test gradients are being measured

continuously but the training gradients are being measured discretely (see Figure 4.1).

To account for this asymmetry in representation, we will redefine our data using an indicator to separate training points from all other points in the input space.

Definition 4.3.3. *Let X be two copies of a Hilbert space H with indices* 0 *and* 1 *so that $X = H \times \{0,1\}$. We will write $x \in H \times \{0,1\}$ so that $x = (x_H, x_I)$ (For brevity, we will omit writing $_H$ and assume each of the following functions defined on H will use x_H and x_I will be a hidden indicator). Let f_w be a differentiable function on H parameterized by $w \in \mathbb{R}^d$. Let $X_T = \{(x_i, 1)\}_{i=1}^M$ be a finite subset of X of size M with corresponding observations $Y_T = \{y_{x_i}\}_{i=1}^M$ with initial parameters w_0 so that there is a constant $b \in \mathbb{R}$ such that for all x, $f_{w_0}(x) = b$. Let L be a differentiable loss function of two values which maps $(f(x), y_x)$ into the positive real numbers. Starting with f_{w_0}, let $\{w_s\}$ be the sequence of points attained by N forward Euler steps of fixed size ε so that $w_{s+1} = w_s - \varepsilon \nabla L(f(X_T), Y_T)$. Let $w_s(t) = w_s + t(w_{s+1} - w_s)$. Let $x \in H \times \{0\}$ be arbitrary and within the domain of f_w for every w. Then $f_{w_s(t)}$ is a* finite parametric gradient model (FPGM).

Definition 4.3.4. *Let $f_{w_s(t)}$ be an FPGM with all corresponding assumptions. Then, for a given training step s, the* exact path kernel *(EPK) can be written*

$$K_{EPK}(x, x', s) = \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x') \rangle dt \tag{4.4}$$

where

$$\phi_{s,t}(x) = \nabla_w f_{w_s(t,x)}(x) \tag{4.5}$$

$$w_s(t) = w_s - t(w_s - w_{s+1}) \tag{4.6}$$

$$w_s(t,x) = \begin{cases} w_s(0), & \text{if } x_I = 1 \\ w_s(t), & \text{if } x_I = 0 \end{cases} \tag{4.7}$$

Note: *ϕ is deciding whether to select a continuously or discrete gradient based on whether the data is from the training or testing copy of the Hilbert space H. This is due to the inherent asymmetry that is apparent from the derivation of this kernel (see Appendix section 4.3.1). This choice avoids potential discontinuity in the kernel output when a test set happens to contain training points.*

Lemma 4.3.5. *The exact path kernel (EPK) is a kernel.*

Proof. We must show that the associated kernel matrix $K_{\text{EPK}} \in \mathbb{R}^{n\times n}$ defined for an arbitrary subset of data $\{x_i\}_{i=1}^M \subset X$ as $K_{\text{EPK},i,j} = \int_0^1 \langle \phi_{s,t}(x_i), \phi_{s,t}(x_j)\rangle dt$ is both symmetric and positive semi-definite.

Since the inner product on a Hilbert space $\langle \cdot, \cdot \rangle$ is symmetric and since the same mapping φ is used on the left and right, K_{EPK} is **symmetric**.

To see that K_{EPK} is **Positive Semi-Definite**, let $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)^\top \in \mathbb{R}^n$ be any vector. We need to show that $\alpha^\top K_{\text{EPK}} \alpha \geq 0$. We have

$$\alpha^\top K_{\text{EPK}}\alpha = \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\int_0^1\langle\phi_{s,t}(x_i),\phi_{s,t}(x_j)\rangle dt \tag{4.8}$$

$$= \int_0^1\left\langle\sum_{i=1}^{n}\alpha_i\phi_{s,t}(x_i),\sum_{j=1}^{n}\alpha_j\phi_{s,t}(x_j)\right\rangle dt \tag{4.9}$$

Re-ordering the sums so that their indices match, we have (4.10)

$$= \int_0^1\left|\sum_{i=1}^{n}\alpha_i\phi_{s,t}(x_i)\right| dt \tag{4.11}$$

$$\geq 0. \tag{4.12}$$

Note that this reordering does not depend on the continuity of our mapping function $\phi_{s,t}(x_i)$.

□

Theorem 4.3.6 (Exact Kernel Ensemble Representation)**.** *A model f_{w_N} trained using discrete steps matching the conditions of the exact path kernel has the following exact representation as an ensemble of N kernel machines:*

$$f_{w_N} = KE(x) := \sum_{s=1}^{N}\sum_{i=1}^{M}a_{i,s}K_{EPK}(x,x_i,s) + b \tag{4.13}$$

where

$$a_{i,s} = -\varepsilon\frac{dL(f_{w_s(0)}(x_i),y_i)}{df_{w_s(0)}(x_i)} \tag{4.14}$$

$$b = f_{w_0}(x) \tag{4.15}$$

Proof. Let f_w be a differentiable function parameterized by parameters w which is trained via N forward Euler steps of fixed step size ε on a training dataset X with labels Y, with initial parameters w_0 so that there is a constant b such that for every x, $f_{w_0}(x) = b$, and weights at each step $w_s : 0 \leq s \leq N$. Let $x \in X$ be arbitrary and within the domain of f_w for every w. For the final trained state of this model f_{w_N}, let $y = f_{w_N}(x)$.

For one step of training, we consider $y_s = f_{w_s(0)}(x)$ and $y_{s+1} = f_{w_{s+1}}(x)$. We wish to account for the change $y_{s+1} - y_s$ in terms of a gradient flow, so we must compute $\dfrac{\partial y}{dt}$ for a continuously varying parameter t. Since f is trained using forward Euler with a step size of $\varepsilon > 0$, this derivative is determined by a step of fixed size of the weights w_s to w_{s+1}. We parameterize this step in terms of the weights:

$$\frac{dw_s(t)}{dt} = (w_{s+1} - w_s) \tag{4.16}$$

$$\int_0^T \frac{dw_s(t)}{dt} dt = \int_0^T (w_{s+1} - w_s) dt \tag{4.17}$$

Since f is being trained using forward Euler, across the entire training set X we can write:

$$\frac{dw_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(X), y_i) = -\varepsilon \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial w} \tag{4.18}$$

For the loss function $L(y, y')$ we will define its derivative with respect to y as $L'(y, y')$. Applying chain rule with this notation and the above substitution, we can write

$$\frac{d\hat{y}}{dt} = \frac{df_{w_s(t)}(x)}{dt} = \sum_{j=1}^{d} \frac{\partial f}{\partial w_j} \frac{dw_j}{dt} \tag{4.19}$$

$$= \sum_{j=1}^{d} \frac{\partial f_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \frac{\partial L(f_{w_s(0)}(X_T), Y_T)}{\partial w_j} \right) \tag{4.20}$$

$$= \sum_{j=1}^{d} \frac{\partial f_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \frac{\partial f_{w_s(0)}(x_i)}{\partial w_j} \right) \tag{4.21}$$

$$= -\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j} \frac{df_{w_s(0)}(x_i)}{\partial w_j} \tag{4.22}$$

$$= -\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{4.23}$$

$$\tag{4.24}$$

Using the fundamental theorem of calculus, we can compute the change in the model's output over step s

$$y_{s+1} - y_s = \int_0^1 -\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) dt \tag{4.25}$$

$$= -\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt \right) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{4.26}$$

$$\tag{4.27}$$

For all N training steps, we have

$$
\begin{aligned}
y_N &= b + \sum_{s=1}^{N} y_{s+1} - y_s \\
y_N &= b + \sum_{s=1}^{N} -\varepsilon \sum_{i=1}^{M} L'(f_{w_s(0)}(x_i), y_i) \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt \right) \cdot \nabla_w f_{w_s(0)}(x_i) \\
&= b + \sum_{i=1}^{M} \sum_{s=1}^{N} -\varepsilon L'(f_{w_s(0)}(x_i), y_i) \int_0^1 \left\langle \nabla_w f_{w_s(t,x)}(x), \nabla_w f_{w_s(t,x_i)}(x_i) \right\rangle dt \\
&= b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt
\end{aligned}
$$

Since an integral of a symmetric positive semi-definite function is still symmetric and positive-definite, each step is thus represented by a kernel machine.

□

Having established this representation, we can introduce $P_S(t)$, the training path which is composed by placing each of the S training steps end-to-end. We can rewrite 4.30 by combining $\sum_{s=1}^{S}$ and $\int_0^1 dt$ into a single integral $\int_{P_S}$:

$$
y_N = b + -\varepsilon \sum_{i=1}^{M} \int_{P_S} L'(f_{w_s(0)}(x_i), y_i) \left(\nabla_w f_{w_s(t)}(x) \right) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{4.28}
$$

$$
\tag{4.29}
$$

Rewriting this way, we can re-evaluate another assumption made during our statement of this theorem, the bias term b. We have forced b to be a constant in order to give our representation a chance of reducing to the form of a kernel machine in a later theorem 5.3.1. Let us relax this and replace b with $f_{w_0(0)}(x)$. We can see that

this new representation no longer requires assumptions about $f_{w_0(0)}(x)$ which gives us a much more general representation which includes most ANNs in practice,

$$f_{w_F(0)}(x) = f_{w_0(0)}(x) + -\varepsilon \sum_{i=1}^{N} \sum_{s}^{S} \int_{P_S} L'(f_{w_s(0)}(x_i), y_i) \langle \nabla_w f_{w_s(t)}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt. \tag{4.30}$$

Remark 1 Note that in this formulation, b depends on the test point x. In order to ensure information is not being leaked from the kernel into this bias term the model f must have constant output for all input. When relaxing this property, to allow for models that have a non-constant starting output, but still requiring b to remain constant, we note that this representation ceases to be exact for all x. The resulting approximate representation has logit error bounded by its initial bias which can be chosen as $b = \text{mean}(f_{w_0(0)}(X_T))$. Starting bias can be minimized by starting with small parameter values which will be out-weighed by contributions from training. In practice, we sidestep this issue by initializing all weights in the final layer to 0, resulting in $b = \log(\text{softmax}(0))$, thus removing b's dependence on x.

Remark 2 The exactness of this proof hinges on the *separate* measurement of how the model's parameters change. The gradients on training data, which are fixed from one step to the next, measure how the parameters are changing. This is opposed to the gradients on test data, which are *not* fixed and vary with time. These measure a continuous gradient field for a given point. We are using interpolation as a way to measure the difference between the step-wise linear training path and the continuous loss gradient field.

Theorem 4.3.7 (Exact Kernel Machine Reduction)**.** *Let* $\nabla L(f(w_s(x), y)$ *be constant*

across steps s, $(a_{i,s}) = (a_{i,0})$*. Let the kernel across all N steps be defined as* $K_{NEPK}(x, x') = \sum_{s=1}^{N} a_{i,0} K_{EPK}(x, x', s)$ *Then the exact kernel ensemble representation for* f_{w_N} *can be reduced exactly to the kernel machine representation:*

$$f_{w_N}(x) = KM(x) := b + \sum_{i=1}^{M} a_{i,0} K_{NEPK}(x, x') \tag{4.31}$$

which is to say that all such models, which are sheaves in the RKBS of functions integrated along discrete optimization paths in fact reduce to a sheaf in the reproducing kernel Hilbert space (RKHS).

See Appendix C.0.1 for full proof. By combining theorems 4.3.6 and 5.3.1, we can construct an exact kernel machine representation for any arbitrary parameterized model trained by gradient descent which satisfies the additional property of having constant loss across training steps (e.g. any ANN using catagorical cross-entropy loss (CCE) for classification). This representation will produce exactly identical output to the model across the model's entire domain. This establishes exact kernel-neural equivalence for classification ANNs. Furthermore, Theorem 4.3.6 establishes an exact kernel ensemble representation without limitation to models using loss functions with constant derivatives across steps. It remains an open problem to determine other conditions under which this ensemble may be reduced to a single kernel representation.

4.3.2 Discussion

The map $\phi_{s,t}(x)$ depends on both s and t, which is non-standard but valid, however an important consequence of this formulation is that the output of this representation is not guaranteed to be continuous. This discontinuity is exactly measuring the error

between the model along the exact path compared with the gradient flow for each step.

We can write another function k' which is continuous but not symmetric, yet still produces an exact representation:

$$k'(x, x') = \langle \nabla_w f_{w_s(t)}(x), \nabla_w f_{w_s(0)}(x') \rangle. \tag{4.32}$$

The resulting function is a valid kernel if and only if for every s and every x,

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt = \nabla_w f_{w_s(0)}(x). \tag{4.33}$$

We note that since f is being trained using forward Euler, we can write:

$$\frac{dw_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(x_i), y_i). \tag{4.34}$$

In other words, our parameterization of this step depends on the step size ε and as $\varepsilon \to 0$, we have

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt \approx \nabla_w f_{w_s(0)}(x). \tag{4.35}$$

In particular, given a model f that admits a Lipshitz constant K this approximation has error bounded by εK and a proof of this convergence is direct. This demonstrates that the asymmetry of this function is exactly measuring the disagreement between the discrete steps taken during training with the gradient field. This function is one of several subjects for further study, particularly in the context of Gaussian processes whereby the asymmetric Gram matrix corresponding with this function can stand in

for a covariance matrix. It may be that the not-symmetric analogue of the covariance in this case has physical meaning relative to uncertainty.

4.3.3 Independence from Optimization Scheme

We can see that by changing equation 4.34 we can produce an exact representation for any first order discrete optimization scheme that can be written in terms of model gradients aggregated across subsets of training data. This could include backward Euler, leapfrog, and any variation of adaptive step sizes. This includes stochastic gradient descent, and other forms of subsampling (for which the training sums need only be taken over each sample). One caveat is adversarial training, whereby the a_i are now sampling a measure over the continuum of adversarial images. We can write this exactly, however computation will require approximation across the measure. Modification of this kernel for higher order optimization schemes remains an open problem.

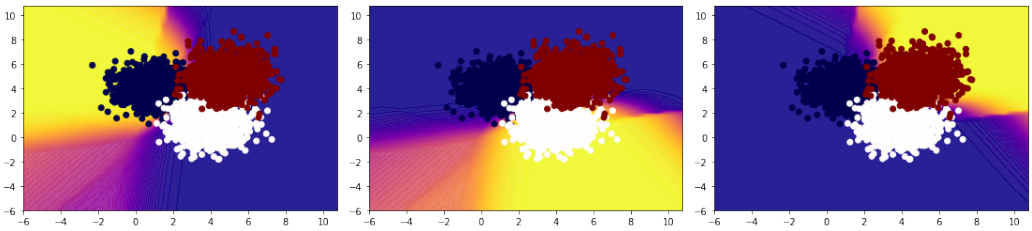


FIGURE 4.3: Updated predictions with kernel a_i updated via gradient descent with training data overlaid for classes 1 (left), 2 (middle), and 3 (right). The high prediction confidence in regions far from training points demonstrates that the learned kernel is non-stationary.

4.3.4 Ensemble Reduction

In order to reduce the ensemble representation of Equation (4.13) to the kernel representation of Equation (5.1), we require that the sum over steps still retain the properties of the kernel (symmetry and positive semi-definiteness). In particular we require that for every subset of the training data x_i and arbitrary α_i and α_j, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{M}\sum_{s=1}^{N}\alpha_i\alpha_j a_{l,s}\int_0^1 K_{\text{EPK}}(x_i,x_j)dt \geq 0 \tag{4.36}$$

A sufficient condition for this reduction is that the gradient of the loss function does not change throughout training. This is the case for categorical cross-entropy where labels are in $\{0,1\}$. In fact, in this specific context the gradient of the loss function does not depend on $f(x)$, and are fully determined by the ground truth label, making the gradient of the cross-entropy loss a constant value throughout training (See Appendix section C.0.1). Showing the positive-definiteness of more general loss functions (e.g. mean squared error loss) will likely require additional regularity conditions on the training path, and is left as future work.

4.3.5 Prior Work

Constant sign loss functions have been previously studied by Chen et al. (2021b), however the kernel that they derive for a finite-width case is of the form

$$K(x,x_i) = \int_0^T |\nabla_f L(f_t(x_i),y_i)|\langle\nabla_w f_t(x),\nabla_w f_t(x_i)\rangle dt \tag{4.37}$$

The summation across these terms satisfies the positive semi-definite requirement of a kernel. However the weight $|\nabla L(f_t(x_i), y_i)|$ depends on x_i which is one of the two inputs. This makes the resulting function $K(x, x_i)$ asymmetric and therefore not a kernel.

4.3.6 Uniqueness

Uniqueness of this kernel is not guaranteed. The mapping from paths in gradient space to kernels is in fact a function, meaning that each finite continuous path has a unique exact kernel representation of the form described above. However, this function is not necessarily onto the set of all possible kernels. This is evident from the existence of kernels for which representation by a finite parametric function is impossible. Nor is this function necessarily one-to-one since there is a continuous manifold of equivalent parameter configurations for neural networks. For a given training path, we can pick another path of equivalent configurations whose gradients will be separated by some constant $\delta > 0$. The resulting kernel evaluation along this alternate path will be exactly equivalent to the first, despite being a unique path. We also note that the linear path l_2 interpolation is not the only valid path between two discrete points in weight space. Following the changes in model weights along a path defined by Manhattan Distance is equally valid and will produce a kernel machine with equivalent outputs. It remains an open problem to compute paths from two different starting points which both satisfy the constant bias condition from Definition (4.3.4) which both converge to the same final parameter configuration and define different kernels.

4.4 Experimental Results

Our first experiments test the kernel formulation on a dataset which can be visualized in 2d. These experiments serve as a sanity check and provide an interpretable representation of what the kernel is learning.

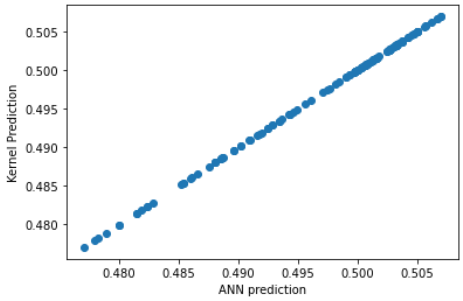


FIGURE 4.4: Class 1 EPK Kernel Prediction (Y) versus neural network prediction (X) for 100 test points, demonstrating extremely close agreement.

4.4.1 Evaluating The Kernel

A small test data set within 100 dimensions is created by generating 1000 random samples with means $(1, 4, 0, ...)$, $(4, 1, 0, ...)$ and $(5, 5, 0, ...)$ and standard deviation 1.0. These points are labeled according to the mean of the Gaussian used to generate them, providing 1000 points each from 3 classes. A fully connected ReLU network with 1 hidden layer is trained using categorical cross-entropy (CCE) and gradient descent with gradients aggregated across the entire training set for each step. We then compute the EPK for this network, approximating the integral from Equation 4.4 with 100 steps which replicates the output from the ReLU network within machine precision. The EPK (Kernel) outputs are compared with neural network predictions in Fig. 4.4 for class 1. Having established this kernel, and its corresponding kernel

machine, one natural extension is to allow the kernel weights a_i to be retrained. We perform this updating of the krenel weights using a SVM and present its predictions for each of three classes in Fig. 4.3.

4.4.2 Kernel Analysis

Having established the efficacy of this kernel for model representation, the next step is to analyze this kernel to understand how it may inform us about the properties of the corresponding model. In practice, it becomes immediately apparent that this kernel lacks typical properties preferred when humans select kernels. Fig. 4.3 show that the weights of this kernel are non-stationary on our toy problem, with very stable model predictions far away from training data. Next, we use this kernel to estimate uncertainty. Consistent with many other research works on Gaussian processes for classification (e.g. Rasmussen, Williams, et al. (2006)) we use a GP to regress to logits. We then use Monte-Carlo to estimate posteriors with respect to probabilities (post-soft-max) for each prediction across a grid spanning the training points of our toy problem. The result is shown on the right-hand column of Fig. 4.5. We can see that the kernel values are more confident (lower standard deviation) and more stable (higher kernel values) the farther they get from the training data in most directions.

In order to further understand how these strange kernel properties come about, we exercise another advantage of a kernel by analyzing the points that are contributing to the kernel value for a variety of test points. In Fig. 4.6 we examine the kernel values for each of the training points during evaluation of three points chosen as the mean of the generating distribution for each class. The most striking property of these kernel point values is the fact that they are not proportional to the euclidean

distance from the test point. This appears to indicate a set of basis vectors relative to each test point learned by the model based on the training data which are used to spatially transform the data in preparation for classification. This may relate to the correspondence between neural networks and maximum margin classifiers discussed in related work (Chizat and Bach, 2020; Shah, Jain, and Netrapalli, 2021). Another more subtle property is that some individual data points, mostly close to decision boundaries are slightly over-weighted compared to the other points in their class. This latter property points to the fact that during the latter period of training, once the network has already achieved high accuracy, only the few points which continue to receive incorrect predictions, i.e. caught on the wrong side of a decision boundary, will continue contributing to the training gradient and therefore to the kernel value.

4.4.3 Extending To Image Data

We perform experiments on MNIST to demonstrate the applicability to image data. This kernel representation was generated for convolutional ReLU Network with the categorical cross-entropy loss function, using Pytorch (Paszke et al., 2019). The model was trained using forward Euler (gradient descent) using gradients generated as a sum over all training data for each step. The state of the model was saved for every training step. In order to compute the per-training-point gradients needed for the kernel representation, the per-input jacobians are computed at execution time in the representation by loading the model for each training step i, computing the jacobians for each training input to compute $\nabla_w f_{w_s(0)}(x_i)$, and then repeating this procedure for 200 t values between 0 and 1 in order to approximate $\int_0^1 f_{w_s(t)}(x)$. For MNIST, the resulting prediction is very sensitive to the accuracy of this integral approximation,

as shown in Fig. 4.7. The top plot shows approximation of the above integral with only one step, which corresponds to the DPK from previous work (Chen et al., 2021b; Domingos, 2020b; Incudini et al., 2022) and as we can see, careful approximation of this integral is necessary to achieve an accurate match between the model and kernel.

4.5 Conclusion and Outlook

The implications of a practical and finite kernel representation for the study of neural networks are profound and yet importantly limited by the networks that they are built from. For most gradient trained models, there is a disconnect between the input space (e.g. images) and the parameter space of a network. Parameters are intrinsically difficult to interpret and much work has been spent building approximate mappings that convert model understanding back into the input space in order to interpret features, sample importance, and other details (Simonyan, Vedaldi, and Zisserman, 2013; Lundberg and Lee, 2017; Selvaraju et al., 2019). The EPK is composed of a direct mapping from the input space into parameter space. This mapping allows for a much deeper understanding of gradient trained models because the internal state of the method has an exact representation mapped from the input space. As we have shown in Fig. 4.6, kernel values derived from gradient methods tell an odd story. We have observed a kernel that picks inputs near decision boundaries to emphasize and derives a spatial transform whose basis vectors depend neither uniformly nor continuously on training points. Although kernel values are linked to sample importance, we have shown that most contributions to the kernel's prediction for a given point are measuring an overall change in the network's internal representation. This supports

the notion that most of what a network is doing is fitting a spatial transform based on a wide aggregation of data, and only doing a trivial calculation to the data once this spatial transform has been determined (Chizat and Bach, 2020). As stated in previous work by Domingos (2020b), this representation has strong implications about the structure of gradient trained models and how they can understand the problems that they solve. Since the kernel weights in this representation are fixed derivatives with respect to the loss function L, $a_{i,s} = -\varepsilon L'(f_{w_s(0)}(x_i), y_i)$, nearly all of the information used by the network is represented by the kernel mapping function and inner product. Inner products are not just measures of distance, they also measure angle. In fact, figure 4.8 shows that for a typical training example, the L_2 norm of the weights changes monotonically by only 20-30% during training. This means that the "learning" of a gradient trained model is dominated by change in angle, which is predicted for kernel methods in high dimensions (Härdle et al., 2004).

For kernel methods, our result also represents a new direction. Despite their firm mathematical foundations, kernel methods have lost ground since the early 2000s because the features implicitly learned by deep neural networks yield better accuracy than any known hand-crafted kernels for complex high-dimensional problems (Bengio, Delalleau, and Roux, 2005). We're hopeful about the scalability of learned kernels based on recent results in scaling kernel methods (Snelson and Ghahramani, 2005). Exact kernel equivalence could allow the use of neural networks to implicitly construct a kernel. This could allow kernel based classifiers to approach the performance of neural networks on complex data. Kernels built in this way may be used with Gaussian processes to allow meaningful direct uncertainty measurement. This would allow for much more significant analysis for out-of-distribution samples including adversarial

attacks (Szegedy et al., 2013; Ilyas et al., 2019). There is significant work to be done in improving the properties of the kernels learned by neural networks for these tools to be used in practice. We are confident that this direct connection between practical neural networks and kernels is a strong first step towards achieving this goal.

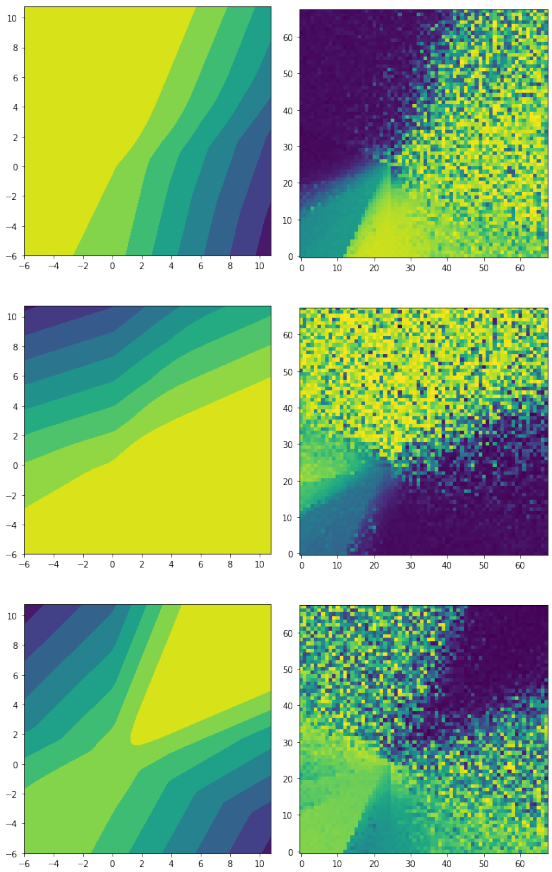


FIGURE 4.5: (left) Kernel values measured on a grid around the training set for our 2D problem. Bright yellow means high kernel value (right) Monte-Carlo estimated standard deviation based on gram matrices generated using our kernel for the same grid as the kernel values. Yellow means high standard deviation, blue means low standard deviation.

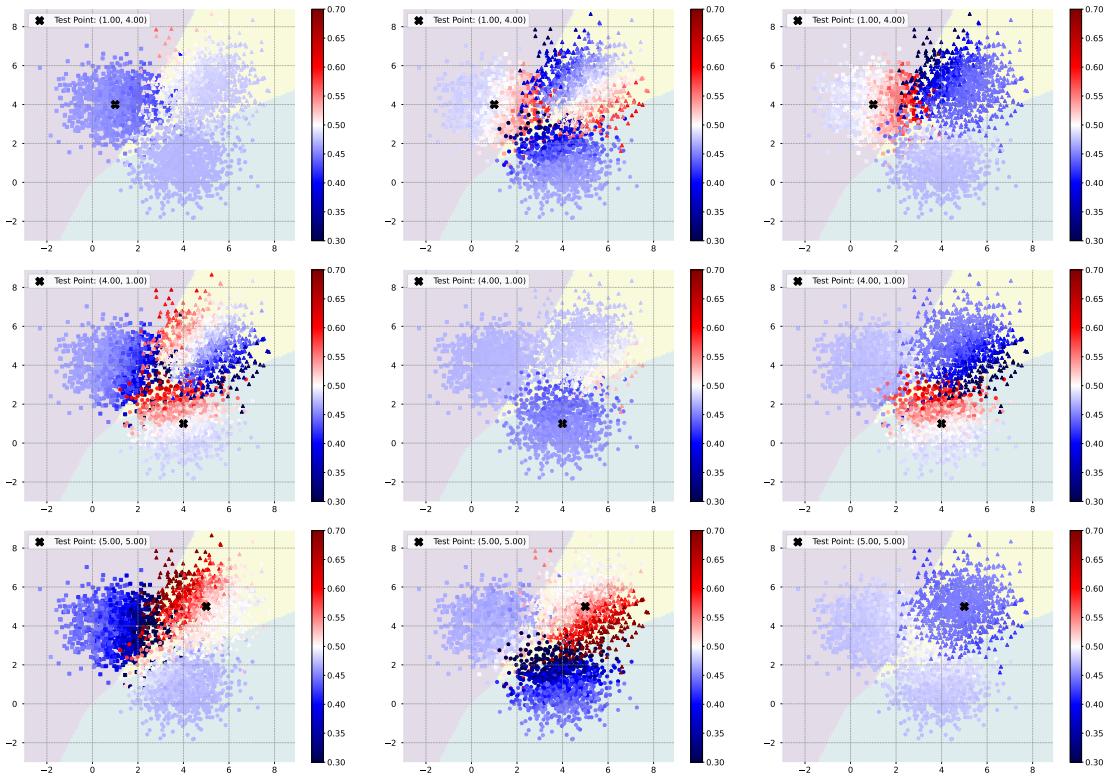
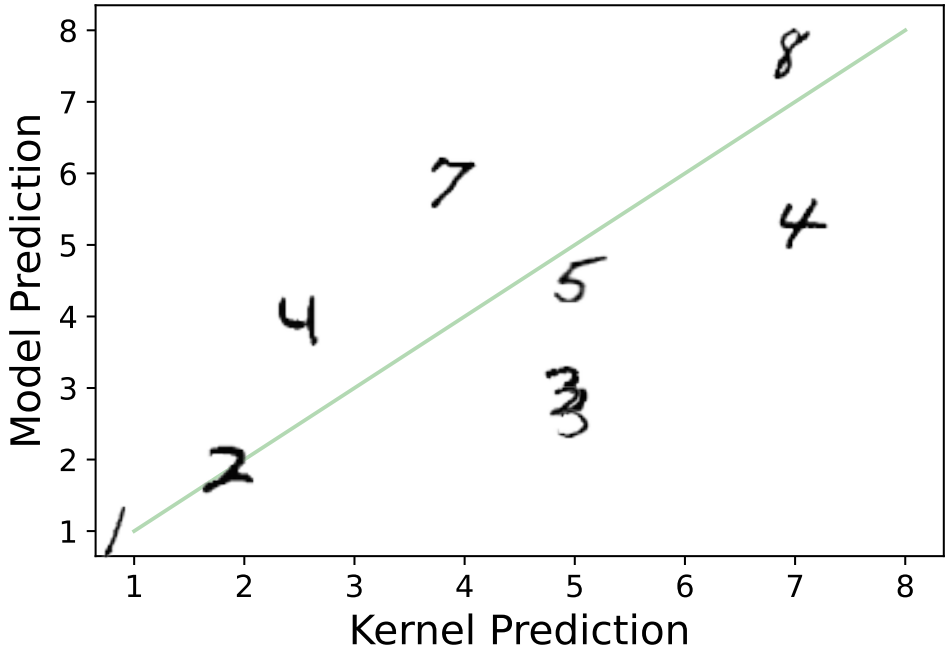
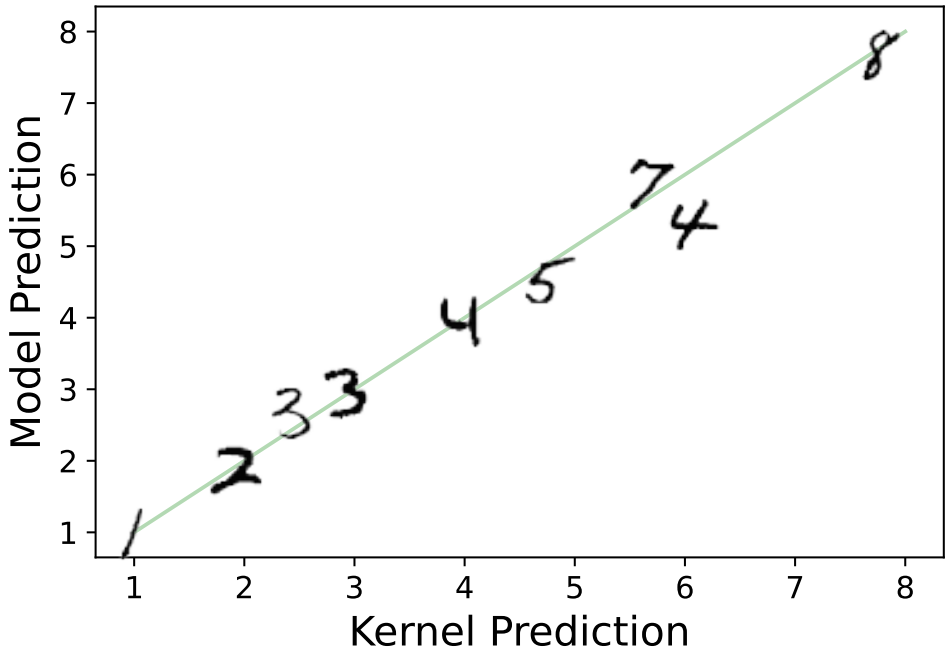
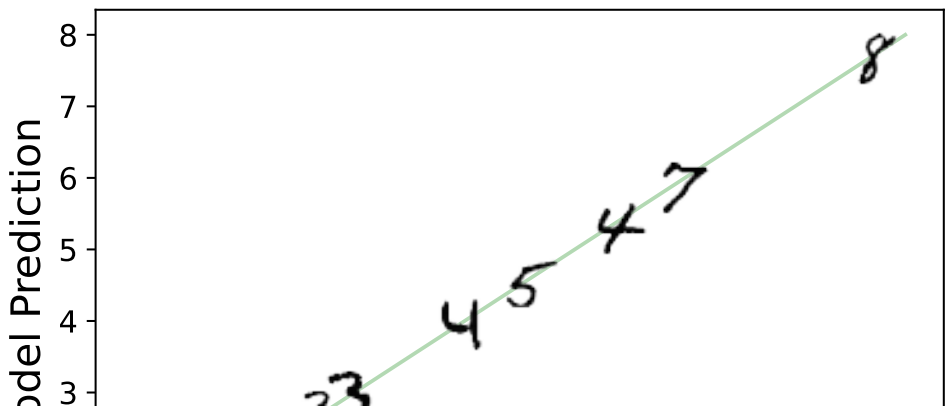


FIGURE 4.6: Plots showing kernel values for each training point relative to a test point. Because our kernel is replicating the output of a network, there are three kernel values per sample on a three class problem. This plot shows kernel values for all three classes across three different test points selected as the mean of the generating distribution. Figures on the diagonal show kernel values of the predicted class. Background shading is the neural network decision boundary.


Model Prediction
Kernel Prediction
1
2
3
4
5
6
7
8


Model Prediction
Kernel Prediction
1
2
3
4
5
6
7
8


odel Prediction
1
2
3
4
5
6
7
8

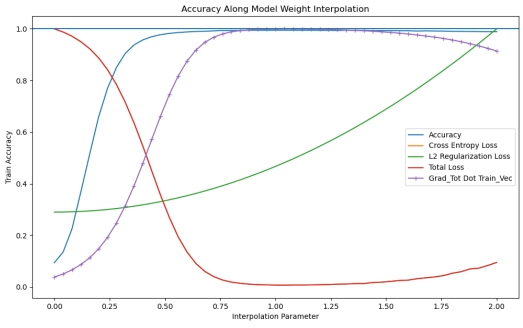


FIGURE 4.8: This plot shows a linear interpolation $w(t) = w_0 + t(w_1 - w_0)$ of model parameters w for a convolutional neural network f_w from their starting random state w_0 to their ending trained state w_1. The hatched purple line shows the dot product of the sum of the gradient over the training data X, $\langle \nabla_w f_{w(t)}(X), (w_1 - w_0)/|w_1 - w_0| \rangle$. The other lines indicate accuracy (blue), total loss (red decreasing), and L2 Regularization (green increasing)

Chapter 5 Exact Path Kernels Naturally Decompose Model Predictions

We can think about the previous paper establishing the exact path kernel as an attempt to determine the necessary conditions for a neural network to be expressed exactly by a kernel machine. The paper also focused on using this kernel representation for uncertainty quantification as an application. Although this is interesting from a theory perspective, the applications are very limited and are a few layers of abstraction away from providing any practical benefit to modern machine-learning techniques. The purpose of the following paper is to expand this theory and develop it towards applications that can be directly useful to the field. In the process of generalizing and applying the above method as a decomposition, it became obvious that one modern machine-learning technique was implicitly relying on this decomposition rather heavily : Out-of-Distribution (OOD) Detection. The problem of identifying OOD data is orthogonal to the adversarial problem. In some sense adversarial problems are difficult because there do not exist practical metrics which can determine that an adversarial example is not part of the natural data distribution for a particular task. For OOD examples, data are generated by distributions which are generally quite distinct from the distribution of training data for an ML model. It can still be time-consuming or difficult using statistical techniques to identify this data, so there is a natural desire to use trained ML models to determine whether data could be samples from the distributions they were trained on or not. In many of the applications that follow

This Chapter includes a paper recently submitted to ICLR 2023. The contents include a generalization of the representation from 4 and two applications of this representation: First to Out-Of-Distribution (OOD) detection, and the second to measuring signal manifold dimension. These two applications begin to showcase the advantages of path kernel ensemble representations of neural networks. This was joint work primarily performed by Brian Bell, Michael Geyer, where most of the theoretical work and mathematics was derived and written by Brian Bell and the experimental work and numerical results were produced by Michael Geyer. This particular paper includes an extensive lit review (performed by Brian Bell) analyzing some methods that have recently come to occupy the cutting-edge of OOD detection algorithms. In the context of this dissertation, this paper includes two important contributions, one is a cleaner general definition of the representation from the previous paper. The other is the decomposition of predictions by taking gradients of this representation with respect to various spaces. This second contribution allows the application of this theory to a class of recent work and demonstrates the ability of this theoretical foundation to inform practical applications at the cutting edge.

5.1 Introduction

Out-of-distribution (OOD) detection for machine learning models is a new, quickly growing field important to both reliability and robustness (Hendrycks and Dietterich, 2019; Biggio et al., 2014; Hendrycks and Gimpel, 2017; Silva et al., 2023; Yang et al., 2021; Filos et al., 2020). Recent results have empirically shown that parameter gradients are highly informative for OOD detection (Behpour et al., 2023; Djurisic

et al., 2023; Huang, Geng, and Li, 2021a). To our knowledge, this paper is the first to present theoretical justifications which explain the surprising effectiveness of parameter gradients for OOD detection.

In this paper, we unite empirical insights in cutting edge OOD with recent theoretical development in the representation of finite neural network models with tangent kernels (Bell et al., 2023; Chen et al., 2021b; Domingos, 2020b). Both of these bodies of work share approaches for decomposing model predictions in terms of parameter gradients. However, the Exact Path Kernel (EPK) (Bell et al., 2023) provides not only rigorous theoretical foundation for the use of this method for OOD, but also naturally defines other decompositions which deepen and expand our understanding of model predictions. The application of this theory is directly connected to recent state of the art OOD detection methods.

In addition, this paper provides a connection between tangent kernel methods and dimension estimation. At the core of this technique is the ability to extract individual training point sensitivities on test predictions. This paper demonstrates a generalization (the gEPK) of the EPK from Bell et al. (2023), which can exactly measure the *input gradient* $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. It is shown that this quantity provides all necessary information for measuring the dimension of the *signal manifold* Srinivas, Bordt, and Lakkaraju (2023) around a given test point.

In short, this work leverages the gEPK to:

- Introduce a theoretical framework that provides a decomposition in terms of the previously proposed EPK.
- Generalize and explain the success of recent successful methods in OOD.

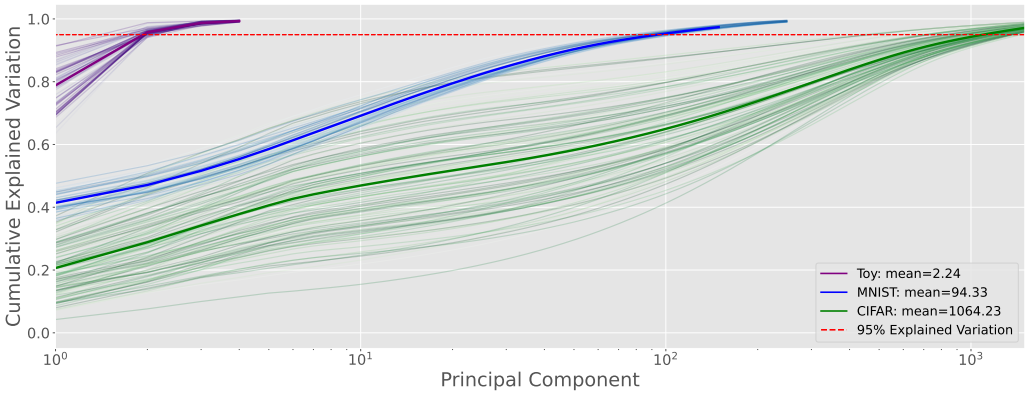


FIGURE 5.1: The gEPK naturally provides a measure of input dimension. This plot shows the CDF of the explained variation of training point sensitivities $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Different datasets are color coded to show differences in signal dimension. Decomposing the input space in this way provides a view of the signal dimension around individual test points. For a toy problem (3 Gaussian distributions embedded in 100 dimensional space) the model only observes between 2 and 3 unique variations which contribute to 95% of the information required for prediction. Meanwhile the dimension of the signal manifold observed by the model around MNIST and CIFAR test points is approximately 94 and 1064 respectively.

- Showcase OOD using natural gEPK based decomposition of model predictions in terms of parameter gradients.

- Measure exact input variations and signal manifold dimension around arbitrary test points.

The primary contributions of this paper are theoretical in nature: establishing the exact representation theorem in Section 5.3 and writing several leading OOD detection methods in terms of this representation. The preliminary experimental results also support practical tasks of out-of-distribution (OOD) detection and estimating signal manifold dimension.

5.2 Related Work

While there has been a significant amount of recent work studying the Neural Tangent Kernel (NTK) (Jacot, Gabriel, and Hongler, 2018), there is still relatively little work exploring its exact counterpart, the path kernels (Bell et al., 2023; Chen et al., 2021b; Domingos, 2020b). While these other works are focused on the precise equivalence between artificial neural networks and SVMs or Kernel machines, this equivalence requires significant restrictions placed on the loss function and model used for a task. This paper seeks to take advantage of this exact representation style without imposing such strict requirements. To the best of our knowledge, this is the first work exploring this loosened equivalence.

There are several schools of thought, whether OOD data can be learned (Huang and Li, 2021; Mohseni et al., 2020; He et al., 2015; Pillai, Fumera, and Roli, 2013; Fumera and Roli, 2002), which part of a model should be interrogated in order to identify OOD examples (Liu et al., 2020; Lin, Roy, and Li, 2021), whether it is a purely statistical question (Lee et al., 2018c), or whether it can simply be solved with more data (Chen et al., 2021a; De Silva et al., 2023). The best performing recent approaches have all used relatively simple modifications of model activation or model gradients (Djurisic et al., 2023; Xu et al., 2023a; Sun and Li, 2022; Sun, Guo, and Li, 2021). The first methods we explore (see Section 5.4 relates to the use of model gradients to construct statistics which separate in-distribution (ID) examples from OOD examples. This is fundamentally a geometric approach which should be comparable with the method proposed by Sun et al. (2022) (Gillette and Kur, 2022) The first prominent method of this type was proposed by Liang, Li, and Srikant (2018). ODIN is still a notable method in this space, and has been followed by many more gradient based

approaches (Behpour et al., 2023; Huang, Geng, and Li, 2021b) and has caused some confusion about why these methods work so well (Igoe et al., 2022)

Much recent work has been devoted to measurement of dimension for the subspace in which the input data distribution live for machine-learning tasks. We will partition this work into works trying to understand this intrinsic data dimension in model agnostic ways (Gillette and Kur, 2022; Yousefzadeh, 2021; Kaufman and Azencot, 2023; Gilmer et al., 2018b; Gong, Boddeti, and Jain, 2019; Glielmo et al., 2022; Facco et al., 2018; Levina and Bickel, 2004) and works trying to understand or extract model's understanding of this subspace (Dominguez-Olmedo et al., 2023; Ansuini et al., 2019; Talwalkar, Kumar, and Rowley, 2008; Costa and Hero, 2004b; Giryes, Plan, and Vershynin, 2014; Zheng et al., 2022). This paper proposes a new method which bears more similarity to the latter. We argue that this approach is more relevant for studying ANNs since they discover their own metric spaces. This paper, additionally, provides tools for exact measurement of the signal manifold around a given test point. Understanding signal manifolds is both useful in practice for more efficient low rank models (Yang et al., 2020b; Swaminathan et al., 2020), and also for uncertainty quantification and robustness (Costa and Hero, 2004a; Wang et al., 2021; Khoury and Hadfield-Menell, 2018; Srinivas, Bordt, and Lakkaraju, 2023; Song et al., 2018; Snoek et al., 2019). Along with OOD, uncertainty quantification is increasingly relying on gradients as demonstrated by Lee and AlRegib (2020) and this paper seeks to also support these methods.

5.3 Theoretical Justification : Generalized Exact Path Kernel

The theoretical foundation of this paper generalizes a recent exact path kernel representation result from Bell et al. (2023). We will reuse the structure of the Exact Path Kernel (EPK) without relying on the reduction to a single kernel across training steps. For classification models trained with cross-entropy loss Bell et al. (2023) showed that the summation over training steps defines a kernel. In order to increase generality, we do not assume the cross-entropy loss, resulting in a representation which is not strictly a kernel. The function, $\varphi_{s,t}(x)$, in the EPK sum defines a bilinear subspace, the properties of which we will study in detail. This representation however, will allow exact and careful decomposition of model predictions according to both input gradients and parameter gradients without the strict requirements of the EPK.

Theorem 5.3.1 (Generalized Exact Path Kernel (gEPK))**.** *Suppose $f(\cdot;\theta)$ is a differentiable parametric model with parameters $\theta \in \mathbb{R}^M$ and L is a loss function. Furthermore, suppose that f has been trained by a series $\{\theta_s\}_{s=0}^{S}$ of discrete steps composed from a sum of loss gradients for the training set $\sum_i^N \varepsilon \nabla_\theta L(f(x_i), y_i)$ on N training data X_T starting from θ_0. Then for an arbitrary test point x, the trained model prediction $f(x;\theta_S)$ can be written:*

$$f(x;\theta_S) = f(x;\theta_0) + \sum_{i=1}^{N}\sum_{s=1}^{S} \varepsilon \left(\int_0^1 \varphi_{s,t}(x)dt\right) \frac{dL(f(x_i,\theta_s), y_i)}{d\hat{y}_{\theta_s(0)}} \left(\varphi_{s,0}(x_i)\right) \tag{5.1}$$

$$\varphi_{s,t}(x) \equiv \nabla_\theta f(x;\theta_s(t)), \tag{5.2}$$

$$\theta_s(t) \equiv \theta_s(0) + t(\theta_{s+1}(0) - \theta_s(0)), \ \textit{and} \tag{5.3}$$

$$\hat{y}_{\theta_s(0)} \equiv f(x;\theta_s(0)). \tag{5.4}$$

Proof. Guided by the proof for Theorem 6 from Bell et al. (2023), let θ and $f(\cdot;\theta)$ satisfy the conditions of Theorem 5.3.1, and x be an arbitrary test point. We will measure the change in prediction during one training step from $\hat{y}_s = f(x;\theta_s)$ to $\hat{y}_{s+1} = f(x;\theta_{s+1})$ according to its differential along the interpolation from θ_s to θ_{s+1}. Since we are training using gradient descent, we can write $\theta_{s+1} \equiv \theta_s + \frac{d\theta_s(t)}{dt}$. We derive a linear interpolate connecting these states:

$$\frac{d\theta_s(t)}{dt} = (\theta_{s+1} - \theta_s) \tag{5.5}$$

$$\int \frac{d\theta_s(t)}{dt} dt = \int (\theta_{s+1} - \theta_s) dt \tag{5.6}$$

$$\theta_s(t) = \theta_s + t(\theta_{s+1} - \theta_s) \tag{5.7}$$

Since f is being trained using a sum of gradients weighted by a constant scalar ε, we can write:

$$\frac{d\theta_s(t)}{dt} = -\varepsilon \nabla_w L(f(X_T;\theta_s(0)), y_i) = -\sum_{i=1}^{N} \varepsilon \sum_{j=1}^{M} \frac{\partial L(f(x_i;\theta_s(0)), y_i)}{\partial \theta^j} \tag{5.8}$$

Applying chain rule and the above substitution, we can write the change in the prediction as

$$\frac{d\hat{y}}{dt} = \frac{df(x;\theta_s(t))}{dt} = \sum_{j=1}^{M} \frac{df}{\partial\theta^j}\frac{\partial\theta^j}{dt} = \sum_{j=1}^{M} \frac{df(x;\theta_s(t))}{\partial\theta^j}\left(-\varepsilon\frac{\partial L(f(X_T,\theta_s(0)),Y_T)}{\partial\theta^j}\right) \tag{5.9}$$

$$= \sum_{j=1}^{M} \frac{df(x;\theta_s(t))}{\partial\theta^j}\left(-\sum_{i=1}^{N}\varepsilon\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\frac{\partial f(x_i;\theta_s(0))}{\partial\theta^j}\right) \tag{5.10}$$

$$= -\sum_{i=1}^{N}\varepsilon\nabla_w f(x;\theta_s(t))\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{5.11}$$

Using the fundamental theorem of calculus, we can compute the change in the model's output over step s,

$$y_{s+1} - y_s = \int_0^1 -\sum_{i=1}^{N}\varepsilon\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\nabla_w f(x;\theta_s(t))\cdot\nabla_w f(x_i;\theta_s(0))dt \tag{5.12}$$

$$= -\sum_{i=1}^{N}\varepsilon\left(\int_0^1\nabla_w f(x;\theta_s(t))dt\right)\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{5.13}$$

For all N training steps, we have

$$y_N = f(x;\theta_0) + \sum_{s=1}^{N} y_{s+1} - y_s \tag{5.14}$$

$$= f(x;\theta_0) - \sum_{s=1}^{N}\sum_{i=1}^{N}\varepsilon\left(\int_0^1\nabla_w f(x;\theta_s(t))dt\right)\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{5.15}$$

□

Remark 1: We note that the training data need not remain fixed for each step of training, the equivalence holds with a different set of training data used at each step, e.g. as in SGD.

Remark 2: This representation holds true for any contiguous subset of a gradient based model, e.g. when applied to only the middle layers of an ANN or only to the final layer. This is since each contiguous subset of an ANN can be treated as an ANN in its own right with the activations of the preceding layer as its inputs and its activations as its outputs. In this case, the training data consisting of previous layer activations may vary as the model evolves.

5.4 OOD is enabled by Parameter Gradients

One natural application of the gEPK is the separation of predictions into vectors corresponding with the test gradient $\varphi_{s,t}(x)$ for a given test point x and each training vector weighted by its loss gradient $\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i)$. While the test vector depends on the choice of test point x, the subspace of training gradient vectors is fixed. By the linear nature of this inner product, it is clear that no variation in test data which is orthogonal to the training vector space can be reflected in a model's prediction. We can say that

$$\left\{\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i); i \in \{1, ..., N\}, s \in \{1, ..., S\}\right\} \tag{5.16}$$

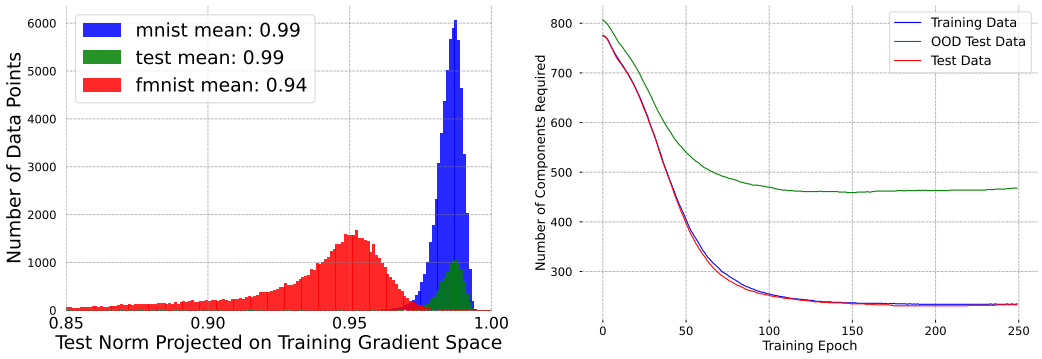


FIGURE 5.2: OOD detection using difference in training vs. test gradients. As the purpose of this paper is not to develop state of the art OOD detection methods, a comparison with recent benchmarks is not provided. Instead, a proof of concept that the gEPK can perform OOD detection is given. Left histogram shows norms of vectors projected onto the gradient weight space defined by the gEPK on MNIST and FMNIST. Right plot shows the number of components required to explain 95% variation in weight space across training for a toy problem (three Gaussian distributions embedded in 100 dimensions).

spans the subspace in which all model predictions can vary. Empirically, we compute the SVD of these training parameter gradients summing over classes in Fig. 5.3. This shows that the space does most of its variation is within only a few dimensions for each test point. Additionally, this corresponds with the well-established notion that the latent dimension of data is much smaller than its ambient dimension. It is by analyzing spectra of this subspace that we may accomplish out-of-distribution (OOD) detection, and, in fact, we will demonstrate that this is how most cutting-edge OOD methods discriminate in practice.

5.4.1 Expressing Prior OOD Methods with the gEPK

We will now establish that most gradient based methods for OOD and some methods which do not explicitly rely on gradients can be written as projections onto subsets of this span.

GradNorm The first well-known method to apply gradient information for OOD is ODIN: Out-of-DIstribution detector for Neural Networks Liang, Li, and Srikant (2018). This method, inspired by adversarial attacks, perturbs inputs by applying perturbations calculated from input gradients. The method then relies on the difference in these perturbations for in-distribution versus out-of-distribution examples to separate these in practice. This method directly inspired Huang, Geng, and Li (2021a) to create GradNorm. This method which occupied the cutting edge in 2021 computes the gradient of Kullback–Leibler divergence with respect to model parameters so that:

$$\frac{1}{C}\sum_{i}^{C}\frac{\partial L_{CE}(f(x;\theta),i)}{\partial \hat{y}}\nabla_{\theta}f(x;\theta) \tag{5.17}$$

This looks like the left side of the inner product from the gEPK, however the scaling factor, $\frac{\partial L_{CE}(f(x;\theta),i)}{d\hat{y}}$, does not match. In fact, this approach is averaging across the parameter gradients of this test point with respect to each of its class outputs, which we can see is only a related subset of the full basis used by the model for predictions. This explains improvements made in later methods that are using a more full basis. Another similar method, ExGrad (Igoe et al., 2022), has been proposed which experiments with different similar decompositions and raises some questions about what is special about gradients in OOD – we hope our result sheds some light on these questions. Another comparable method proposed by Sun et al. (2022) may

also be equivalent through the connection we establish below in Section 5.1 between this decomposition and input gradients which may relate with mapping data manifolds in the Voronoi/Delaunay (Gillette and Kur, 2022) sense.

ReAct, DICE, ASH, and VRA Along with other recent work (Sun, Guo, and Li, 2021; Sun and Li, 2022; Xu et al., 2023a), some of the cutting edge for OOD as of early 2023 involves activation truncation techniques like that neatly described by Djurisic et al. (2023). Given a model, $f(x;\theta) = f^{\text{extract}}(\cdot;\theta_{\text{extract}}) \circ f^{\text{represent}}(\cdot;\theta_{\text{represent}}) \circ f^{\text{classify}}(\cdot;\theta_{\text{classify}})$, and an input, x, a prediction, $f(x,\theta)$, is computed forward through the network. This yields a vector of activations, $A(x,\theta_{\text{represent}})$, in the representation layer of the network. This representation is then pruned down to the p^{th} percentile by setting any activations below that percentile to zero. Djurisic et al. (2023) mention that ASH does not depend on statistics from the training data, however by chain rule, high activations will correspond with high parameter gradients. That means that this truncation is picking a representation for which $\left\langle \nabla_\theta f(x,\theta_{\text{represent}}), \frac{dL(\hat{y}(x_i),y_i)}{d\hat{y}} \nabla_\theta f(x_i,\theta_{\text{represent}}) \right\rangle$ is high for many training data, x_i. This is effectively a projection onto the parameter tangent space of the training data with the highest variation. This may explain some part of the performance advantage of these method.

GradOrth Behpour et al. (2023) explicitly create a reference basis from parameter gradients on training data for comparison. They do this for only the last layer of a network with mean squared error (MSE) loss, allowing a nicely abbreviated expression for the gradient:

$$\nabla_\theta L(x,y) = (\theta x - y)x^T = \Omega x^T \tag{5.18}$$

Treating Ω as an error vector, they prove that all variation of the output must be

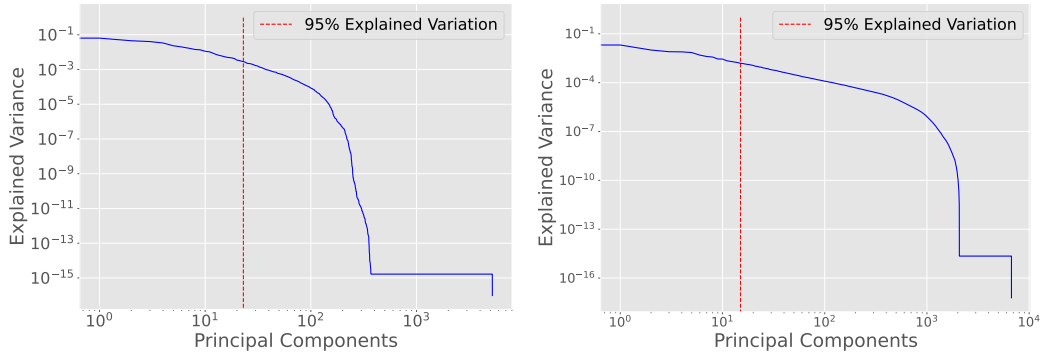


FIGURE 5.3: Explained Variance Ratio of parameter gradients. Left: MNIST, Right: CIFAR. 95% of variation can be explained with a relatively low number of components in both cases.

within the span of the x^T over the training set. They then pick a small subset of the training data and record its activations $R^L_{ID} = [x_1, x_2, ..., x_n]$ over which they compute the SVD, $U^L_{ID}\Sigma^L_{ID}(V^L_{ID})^T = R^L_{ID}$. This representation is then truncated to k principal components according to a threshold ϵ_{th} such that

$$\left\| U^L_{ID}\Sigma^L_{ID,k}(V^L_{ID})^T \right\|^2_F \geq \epsilon_{\text{th}} \|R^L_I D\|^2_F. \tag{5.19}$$

This basis $S^L = (U^L_I D)_k$ is now treated as the reference space onto which test points' final layer gradients can be projected. Their score is

$$O(x) = (\nabla_{\theta_L} \mathcal{L}(f(x, \theta_L), y)) S^L (S^L)^T \tag{5.20}$$

We note that this formulation requires a label y for each of the data being tested for inclusion in the data distribution. Despite this drawback, the performance presented by Behpour et al. (2023) is impressive. Behpour et al. (2023) also present a brief proof that gradient updates for the final layer parameters θ_L for an optimization step defined

by a batch of training data must be within the span of the final layer activation for that batch. While this is true, it may greatly underestimate the true rank of the data and also may not necessarily map the correct magnitude of variations along each of the basis directions.

5.4.2 gEPK for OOD

The gEPK provides a more general spanning result immediately. Indeed, the network's prediction can only be influenced by $\{\varphi_{s,0}(x_i)\}$ for the training data batched for each step, s. Loss in the gEPK only appears for the training points. This means that $\varphi_{s,t}(x) = \nabla_\theta f(x; \theta_s(t))$ can be computed for any test point without need for a label. Then all parameter gradients must live in:

$$\text{span}\left(\left\{\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i) : x_i \in X_T, 0 \leq s \leq S\right\}\right). \tag{5.21}$$

We can see that most, if not all, of the above methods can be represented by some set of averaging assumptions on the representation provided by the gEPK. We test the ability of the gEPK to perform OOD detection by direct projection onto the full parameter gradient space spanned by its training parameter gradients using a sum over the class outputs in Fig. 5.2. Indeed, the gEPK helps explain the high performance of gradient based methods due to the implicit inclusion of the training parameter space in model predictions. This serves to illuminate the otherwise confusing discrepancy raised by Igoe et al. (2022).

In addition, we can see that comparison of loss gradients is unnecessary, which allows testing on data without ground truth labels. For most applications, the SVD of

the parameter gradients over all of the training steps and batches can be pre-computed and compared with test points as needed, although as we can see from this body of work, many simplifying assumptions can be made which will preserve the essential bases needed for performance, but still drastically reduce computational cost. Bottom line: It is not necessarily sufficient to pick a basis that spans a target subspace and then truncate based on its variations. The variations must be accurately measured with correct scaling and, as we can see from the gEPK, implicitly depends on *all* parameter states of the model during training.

5.5 Signal Manifold Dimension Estimated with Training Input Gradients

There has been significant interest in quantifying the intrinsic dimension of data Levina and Bickel (2004), Talwalkar, Kumar, and Rowley (2008), Ceruti et al. (2012), Gong, Boddeti, and Jain (2019), and Zheng et al. (2022). Many such techniques are used to estimate the dimension of intrinsic data manifolds, but do not provide a means to define the manifold itself. Using the gEPK, it is possible to measure the dimension of signal manifolds observed by the model, and view the exact basis that form the signal manifold around each point. It is important to note that the gEPK gradient subspaces do not measure the intrinsic dimension of the data. Rather, they reflect the dimension of the signal manifold observed by a particular model. Models which have better representations of data will have a signal manifold more similar to the intrinsic data manifold.

In order to understand the subspace on which a model is sensitive to variation, we may take gradients decomposed into each of the training data. Take, for example, a model, $f(x, \theta)$, which satisfies the necessary conditions for expression as:

$$f(x, \theta_{\text{trained}}) = f(x, \theta_0(0)) + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) dt \tag{5.22}$$

$$\varphi_{s,t}(x) = \nabla_\theta f(x, \theta_s(t)) \tag{5.23}$$

And $\theta_s(t)$ are the parameters of f for training step s and time t so that $\sum_s \int_0^1 \theta_s(t) dt$ integrates the entire training path taken by the model during training. Given a test point x, we can evaluate its subspace by taking, for each x_i:

$$\frac{df(x, \theta_{\text{trained}})}{dx_j} = \frac{df(x, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d\left(\varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{5.24}$$

$$= 0 + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{d\left(\frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{5.25}$$

$$= \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) dt \left(\frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \tag{5.26}$$

We can see that these gradients will be zero except when $i = j$, thus we may summarize these gradients as a matrix (tensor in the multi-class case), G, with

$$G_j = \sum_s \int_0^1 \varphi_{s,t}(x) dt \left(\frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \tag{5.27}$$

While written in this form, it appears we must keep second-order derivatives, however we note that the inner product with $\phi_{s,t}(x)$ eliminates these extra dimensions, so that clever implementation still only requires storage of vectors (low rank matrices in the multi-class case).

The rank of G represents the dimension of the subspace on which the model perceives a test point, x, to live, and we can get more detailed information about the variation explained by the span of this matrix by taking its singular value decomposition (SVD). We can exactly measure the variation explained by each orthogonal component of the span(G) with respect to the given test point x. $G(x)$ can be defined as a map from x to the subspace perceived by the model around x. Any local variations in the input space which do not lie on the subspace spanned by $G(x)$ can not be perceived by the model, and will have no effect on the models output.

On MNIST, $G(x)$ creates a matrix which is of size 60000×784 (training points $\times$ input dimension). This matrix represents the exact measure of each training points contribution towards a given test prediction. Of note is that in practice this matrix is full rank on the input space as seen in Figure 5.4. This is despite MNIST having significantly less degrees of variation than its total input size (many pixels in input space are always 0). Figure 5.1 demonstrates that accounting for 95% of the variation requires only 94 (12%) of the 784 components on average. Similarly, on CIFAR accounting for 95% of explained variation requires 1064 (34%) of the 3096 components. It is likely that different training techniques will provide significantly different signal manifolds and consequently different numbers of components. Examining these effects may lead to deeper understanding of optimization techniques.

We can also examine this subspace with less granularity by taking the parameter

gradients for each training point from its trained state. This involves using each training point as a test point.

$$\frac{df(x_j, \theta_{\text{trained}})}{dx_j} = \frac{df(x_j, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d\left(\varphi_{s,t}(x_j) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{5.28}$$

The left hand side is computable without path-decomposition and so can be computed for each training datum to create a gradient matrix, $H_{\theta_{\text{trained}}}$. Another term, $\frac{df(x_j, \theta_0(0))}{dx_j}$ is also easily computable, yielding another matrix H_{θ_0}. By comparing the rank and span of $H_{\theta_{\text{trained}}}$ and H_{θ_0} we can understand to what extent the model's spatial representation of the data is due to the initial parameter selection and how much is due to the training path. Also, $H_{\theta_{\text{trained}}}$ provides sample of gradients across all training data, which in some sense must be spanned by the model's implicit subspace basis. Despite missing the granular subspace information, the rank of this gradient matrix and its explained variation computed using SVD should be related to the model's implicit subspace rank. It should be noted that while there is a direct relationship between a models variations in input space and weight space, Figure 5.5 shows that this mapping changes greatly from the beginning to end of training and that this spectrum starts out wide (high dimensional) for θ_0 and much more focused (low dimensional) for θ_T.

One interesting property of using input gradients for training data decomposed according to Eq. 5.27 is the ability to compare input gradients across models with different initial parameters and even different architectures. Figure 5.4 demonstrates that two models with different random initializations which have been trained on

the same dataset have a signal manifold which shares many components. This is a known result that has been explored in deep learning through properties of adversarial transferability Szegedy et al. (2013). This demonstrates that the gEPK is capable of measuring the degree to which two models rely on the same features directly. This discovery may lead to the construction of models which are provably robust against transfer attacks.

5.6 Conclusion

This paper presented a general exact path kernel representation for neural networks with a natural decomposition that connects existing out-of-distribution detection methods to a theoretical baseline. This same representation reveals additional connections to dimension estimation and adversarial transferability. These connections are demonstrated with experimental results on computer vision datasets. The key insights provided by this decomposition in this work are that model predictions implicitly depend on the parameter tangent space on its training data and that this dependence enables decomposition relative to a single test point by either parameter gradients, or training input gradients. This allows users to connect how neural networks learn at training time with how each training point influences the final decisions of a network.

This method has many theoretical connections to continuing work in this area including better understanding of how models depend on implicit prior distributions following (e.g. Nagler (2023)), supporting more robust statistical learning under distribution shifts (e.g. Simchowitz et al. (2023)), and supporting more robust learning

by connecting with a growing body of work applying gradients to OOD detection and more generally by connecting to recent results by Silva et al. (2023) which indicate that understanding OOD data can make models more general.

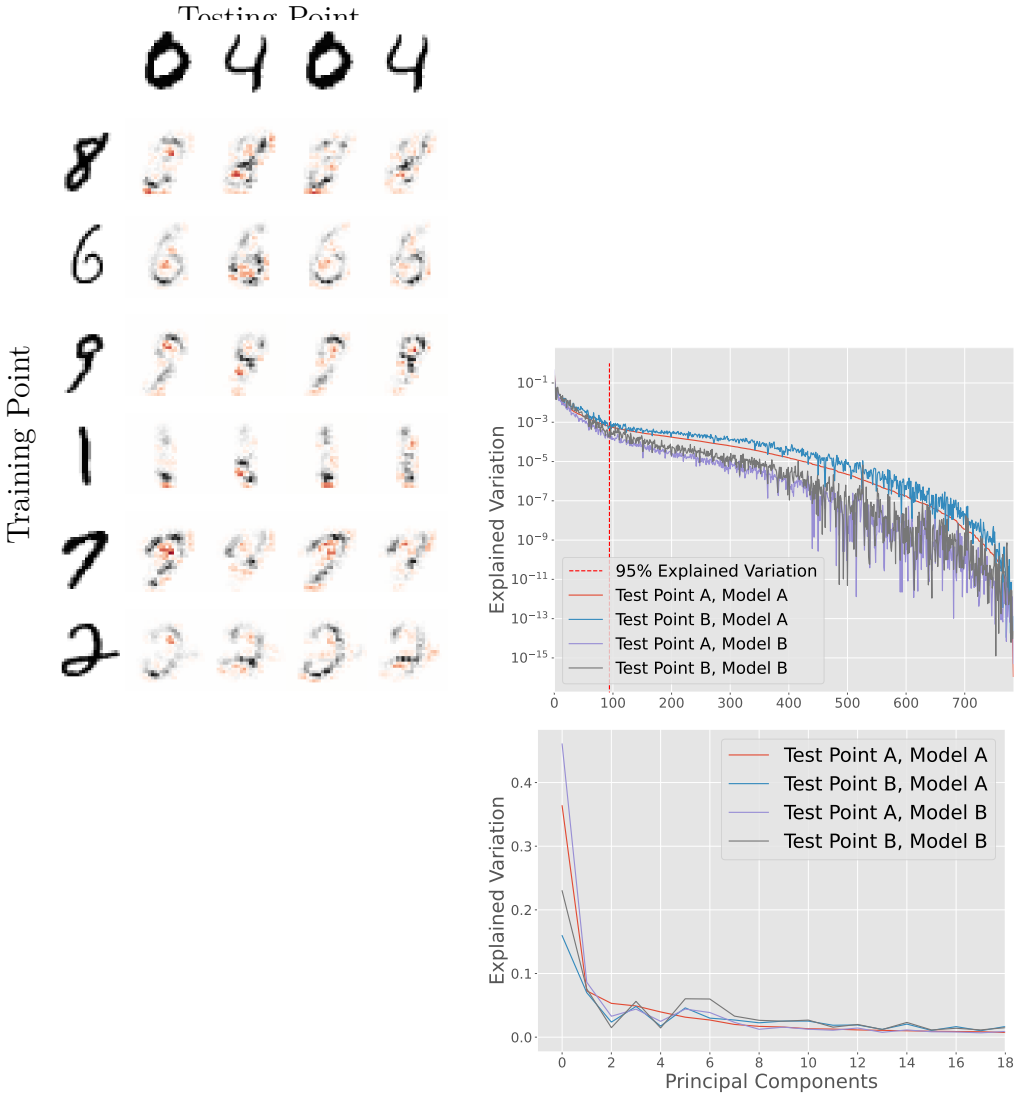


FIGURE 5.4: Left: Visualization of training point input gradients on test points compared between two models. Positive contribution (black) and negative contribution (red) of each training datum to the prediction for each test point. Elements in the grid are $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Right: By taking these individual gradient contributions for a test point and computing the SVD across the training, the significant modes of variation in the input space can be measured (sigma squared). Top is log scale of the full spectrum, bottom shows the first 10 components. Note that this decomposition selects similar, but not identical, modes of variation across test points and even across different models. Components in SVD plots are sorted using Test Point A on Model A.

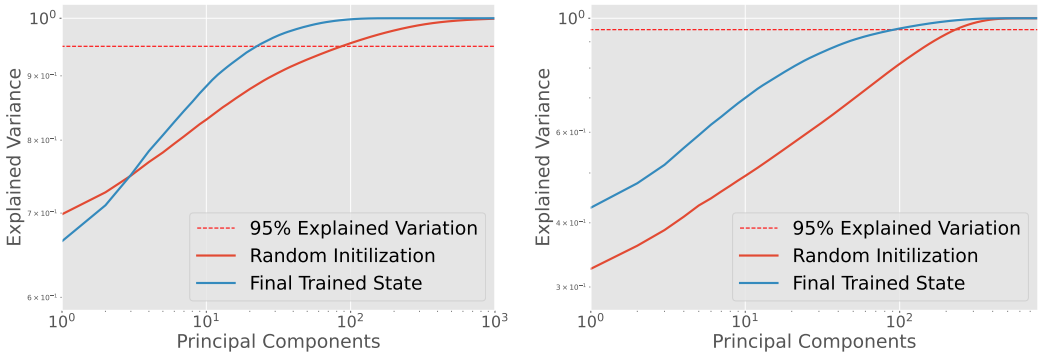


FIGURE 5.5: Differences between observing gradients in input space vs. weight space. Left: CDF of explained variation parameter space. Right: CDF of explained variation input space. Red solid line indicates a model at random initialization while the blue solid line represents the fully trained state. From random initialization, the number of principal components required to achieve 95% explained variation decreases in both cases. Note that at random initialization, the weight space gradients already have only a few directions accounting for significant variation. Disentangling the data dimension using weight space gradients is less effective than doing so in input space (Shamir, Melamed, and BenShmuel, 2021).

Chapter 6 Conclusions and Outlook

Throughout this work, I have sought to understand the intrinsic structure of machine-learning models and how this gives rise to adversarial attacks. I began by studying the construction of neural networks in Chapter. 1 and learning in practice by generating adversarial attacks in Chapter. 2. This led naturally to a more careful study of decision boundaries and the development of the Persistence metric in Chapter 3. 3. Uncanny drops in persistence while crossing decision boundaries toward adversarial attacks indicated that attacks may exist in highly curved regions. The field conspicuously lacks tools for evaluating curvature, although some progress is being made. In the process of lit review for methods that would allow more direct analysis I discovered deficiencies in nascent literature on path kernels starting with the work of Domingos (2020b). The initial work to correct these deficiencies and produce an exact path kernel are presented in Chapter 4. This new exact path kernel representation for neural networks has a primary advantage which is to decompose model predictions into contributions from each training point. This decomposition is in the form of decomposing a prediction in a tangent space, where the exact path kernel implicitly maps the training data into a given tangent space. This mapping can be done for parameter gradients, input space gradients, and several dual tangent spaces related to both of these. Two such decompositions are applied in Chapter. 5 to demonstrate that many cutting edge OOD detection algorithms implicitly use this decomposition in terms of parameter gradients, and that the dual of the input gradients can be used to measure signal manifold dimension.

We can summarize the first 3 chapters as posing some fundamental geometric questions related to machine-learning robustness. Chapters 4 and 5 as propose a new framework for analyzing geometric properties and demonstrating that this framework can be applied very generally. As usual in mathematics, trying to solve a specific problem about geometric causes for adversarial examples has led to a very general result in a totally different field. Although chapters 4 and 5 represent significant progress in understanding ANNs mathematically, they do not directly address the core question that arose within the first three chapters: Is relative dimension and curvature a primary cause of adversarial vulnerability.

Based on this work, we can now make some supportable conjectures:

1. Modern neural network architectures learn decision boundaries that are askew from many interpolations among training and testing data.

2. Non-orthogonal decision boundary crossings indicate that "sharp" corners in the decision space learned by ANN models lead to adversarial examples.

3. Bilinear map based representations allow decomposition of predictions based on training inputs. These show that models implicitly use a lower dimensional implicit representation of data to make predictions.

4. Combining these notions, we have a clear path forward: Decompose adversarial attacks using a bilinear map representation and then compare these signatures of variation with typical modes of variation within training data.

5. Likely this will show that adversarial attacks are arising from sharp corners in decision space, some of which can be mitigated by making models more aware

of the properties of the decision space they have learned, some of which cannot be mitigated because training data are insufficient to fully constrain decision surfaces.

From here, there are several important directions for future research. First is the application of these methods directly to the adversarial robustness questions from Chapters 1-3. Second is the complete generalization of this new framework including conditions under which such representations live in Banach spaces or Hilbert spaces and what order of accuracy can be maintained using truncated spectral decompositions. Third is the application of this theory more broadly to the wide variety of spatial problems that are present in machine-learning.

6.0.1 Applications to Adversarial Robustness

In Fig. 6.1 we can see that sharp geometric curvature and shallow angles are observed when interpolating across decision boundaries between natural and especially adversarial images. One line of research will follow the direct geometric approach by constructing test objects (wedges) which replicate the structure observed in the practical networks. an example of this is shown in 6.1. The second approach is to take advantage of the framework from Chapters 4 and 5 in order to decompose the training gradients at points on the decision boundary to understand neural networks' learned degrees of freedom at these locations. The goal of this line of research is to understand these geometric constraints and eventually pose both updated training objectives and also better definitions for the identification of adversarial examples in practice. This line of research may have implications beyond robustness, to include

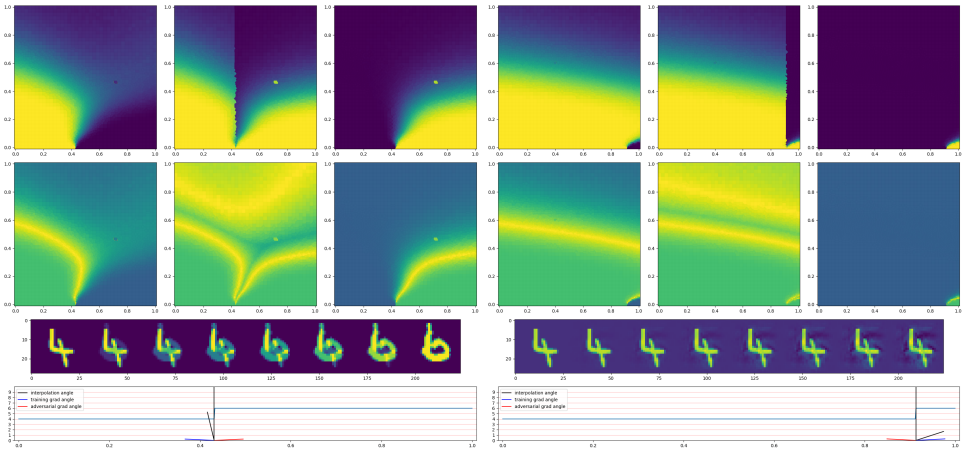


FIGURE 6.1: Left: Interpolation across the decision boundary between two natural images from the test data for MNIST. Right : Interpolation across the decision bundary between a natural example (left) and an adversarial example (right). Top Heatmaps show the fraction of uniform samples from a sphere which return the same class, middle heatmaps show the 0.7 level set of these fractions, bottom plot shows incident angles on the decision boundary of the interpolant (black), the training gradient (blue) and the testing gradient (red).

uncertainty quantification, generalization, out-of-distribution detection, and other useful metrics to create trustworthy AI.

In particular, we can decompose adversarial gradients into an initial gradient plus a component from each training input,

$$f(x,\theta_F) = f(x,\theta_0) + \sum_i \sum_s \int_0^1 \langle \nabla_\theta f(x,\theta_s(t)), f(x_i,\theta_s(t))\rangle dt \tag{6.1}$$

$$\frac{df(x,\theta_F)}{dx} = \frac{df(x,\theta_0)}{dx} + \sum_i \sum_s \int_0^1 \langle \nabla_\theta \frac{df(x,\theta_s(t))}{dx}, f(x_i,\theta_s(t))\rangle dt. \tag{6.2}$$

From this decomposition, we can also replace the individual training data with an orthogonal basis by choosing vectors as in Halko, Martinsson, and Tropp (2011), solving SVD for each step: Take a model, $f(x,\theta)$, which satisfies the necessary

conditions for expression as:

$$f(x,\theta_{\text{tr}}) = f(x,\theta_0(0)) + \sum_s \int_0^1 \left\langle \varphi_{s,t}(x), \sum_i \left(L'(x_i,y_i)\varphi_{s,0}(x_i) \right) \right\rangle dt \tag{6.3}$$

$$\varphi_{s,t}(x) = \nabla_\theta f(x,\theta_s(t)) \tag{6.4}$$

Then we will let the RHS be rows of a matrix $L \cdot A$ where X is a vector with elements $\varphi_{s,t}(x)$, L =diag$(L'(x_i,y_i))$ (size is $[N,N]$), and each row $A_i = \varphi_{s,0}(x_i)$ so that A has size $[N,P]$. Then we can compute $U\Sigma V^T = A$ (U of size $[N,P]$, σ of size $[N,N]$, and V^T of size $[P,P]$. Let's first try projecting each A_i onto the row space of V so that

$$A_i = \sum_k A_i V_k^T V_k \tag{6.5}$$

$$\sum_i L_i A_i = \sum_i \sum_k L_i A_i V_k^T V_k \tag{6.6}$$

$$\sum_i L_i A_i = \sum_k \left(\sum_i L_i A_i V_k^T \right) V_k \tag{6.7}$$

$$\sum_i L_i A_i = \sum_k \left((\vec{1} L A V^T)_k \right) V_k \tag{6.8}$$

$$\tag{6.9}$$

If we truncate V down to K dimensions, then we have $B = \vec{1}LA(V^K)^T$ with sizes $[1,N],[N,N][N,P],[P,K]$ and I can rewrite the truncated representation:

$$\sum_i L_i A_i = \sum_k \left((\vec{1} L A (V^K)^T)_k \right) V_k^K \tag{6.10}$$

$$= \sum_k B_k V_k^K \tag{6.11}$$

$$\tag{6.12}$$

Now, we can rewrite our original expression:

$$f(x, \theta_{\text{tr}}) = f(x, \theta_0(0)) + \sum_s \int_0^1 \left\langle \varphi_{s,t}(x), \sum_k B_k V_k^K \right\rangle dt \tag{6.13}$$

More generally, we can define spectral components across all training steps and perform more general spectral decomposition with further efficiency reduction as in Tancik et al. (2020). The path kernel formulation allows easy management of numerical error for any of these approaches.

Regardless of how sophisticated we make this decomposition, all of these methods have the advantage of maintaining an exact mapping to the tangent space around each input point. This mapping implicitly defines natural dimension reduction – by truncating the chosen basis. The tangent space can be thought of as the models implicit justifications – which training influences are affecting its decisions on this prediction. These directions are implicitly how we can analyze (and potentially penalize) points of high curvature e.g. at the decision boundary. This provides a theoretical foundation from which recent work in manifold study and data augmentation (Kaufman and Azencot, 2023; Liu, He, and Tsai, 2023; Šípka et al., 2023; Cha and Thiyagalingam, 2023; Marbut, Mckinney-Bock, and Wheeler, 2023; Gao et al., 2023; Oh and Yun,

2023; Chen et al., 2023).

6.0.2 Generalization in the sense of Reproducing Kernel Banach Spaces

Given that the representation from Chapters 4 and 5 have a small asymmetry, a general description of these representations cannot fit within Hilbert Space. There is a convenient approach building on the theory of Reproducing Kernel Banach Spaces (RKBS) which is summarized nicely by Zhang, Xu, and Zhang (2009). It is by careful construction of a semi-inner product that I believe our representation can be written in this way. This allows access to tools built for Banach spaces for analysis of both accuracy, risk minimization, and other useful results. A similar line of work is already being pursued by Shilton et al. (2023) with which our work can likely be connected. I believe this is the most interesting natural direction for continuing work. Although this is less likely to produce practical payoffs immediately, I believe that this approach will greatly enhance the theoretical foundation upon which analysis of neural network performance and limitations are based.

6.0.3 Connecting Distributional Learning with Neural Networks

The neural representations and decompositions proposed in this work provide images of the tangent space according to the a given model for each point in the data space. The tangent space for each datum must be connected within the implicit metric space of the neural network, helping us to pose constraints on the geodesics connecting data. It has been shown increasingly in recent work (e.g. by Lu and Lu (2020),

Yang, Li, and Wang (2022), Altekrüger, Hertrich, and Steidl (2023)) that Neural Networks learn distributions in a sense that approximates the Wasserstein metric to some order. Also, work by Chizat and Bach (2020) that neural network classifiers are approximately max-margin classifiers in some implicit space. We cannot necessarily compute this space exactly, however by examining the tangent spaces exposed by the kernel type decomposition, we can pose questions about this metric space in the dual sense. The connection of these three concepts into an understanding of how Neural Networks embed an approximation of the Wasserstein metric into some implicit spaces that is likely euclidean will have significant impact on the machine-learning community at large. This method also serves to connect the prior work from Chapters 1-3 with this later work, examining how models connect data geometrically.

Appendix A Attacks

A.1 L-BFGS

Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) is a quasi-newton gradient based optimization algorithm which stores a history of gradients and positions from each previous optimization step Liu and Nocedal, 1989. The algorithm as implemented to optimize a function f with gradient at step k of g_k is as follows

L-BFGS

Choose $x_0, m, 0 < \beta' < 1/2, \beta' < \beta < 1$, and a symmetric positive definite starting matrix H_0.

for $k = 0$ to $k =$ (the number of iterations so far) **do**

$d_k = -H_k g_k$,

$x_{k+1} = x_k + \alpha_k d_k$, Where α_k satisfies

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \beta' \alpha_k g_k^T d_k,$$
$$g(x_k + \alpha_k d_k)^T d_k \geq \beta g_k^T d_k.$$

▷ Trying steplength $\alpha_k = 1$ first.

Let $\hat{m} = \min(k, m-1)$.

for i from 0 to $\hat{m} + 1$ **do** ▷ Update H_0 $\hat{m} + 1$ times using pairs $\{y_j, s_j\}_{j=k-\hat{m}}^{k}$,

$$\begin{aligned} H_{k+1} = {} & (V_k^T \cdot V_{k-\hat{m}}^T) H_0 (V_{k-\hat{m}} \cdots V_k) \\ & + \rho_{k-\hat{m}} (V_k^T \cdots V_{k-\hat{m}+1}^T) s_{k-\hat{m}} s_{k-\hat{m}}^T (V_{k-\hat{m}+1} \cdots V_k) \\ & + \rho_{k-\hat{m}+1} (V_k^T \cdots V_{k-\hat{m}+2}^T) s_{k-\hat{m}+1} s_{k-\hat{m}+1}^T (V_{k-\hat{m}+2} \cdots V_k) \\ & \vdots \\ & + \rho_k s_k s_k^T \end{aligned}$$

end for

end for

Appendix B Persistence Tools

B.1 Bracketing Algorithm

This algorithm was implemented in Python for the experiments presented.

B.2 Bracketing Algorithm

The Bracketing Algorithm is a way to determine persistance of an image with respect to a given classifier, typically a DNN. The algorithm was implemented in Python for the experiments presented. The RANGEFINDER function is not strictly necessary, in that one could directly specify values of $\sigma_{\min}$ and $\sigma_{\max}$, but we include it here so that the code could be automated by a user if so desired.

B.3 Convolutional neural networks used

In Table 3.1 we reported results on varying complexity convolutional neural networks. These networks consist of a composition of convolutional layers followed by a maxpool and fully connected layers. The details of the network layers are described in Table B.1 where Ch is the number of channels in the convolutional components.

```
function BRACKETING(image, ANN, n, tol, n_real, c_i)        ▷ start with same magnitude noise as image
    u_tol, l_tol = 1.01, 0.99
    a_var = Variance(image)/4                                ▷ Running Variance
    l_var, u_var = 0, a_var*2   ▷ Upper and Lower Variance of search space   ▷ Adversarial image plus noise counts
    a_counts = zeros(n)
    n_sz = image.shape[0]
    mean = Zeros(n_sz)
    I = Identity(n_sz)
    count = 0                ▷ grab the classification of the image under the network
    y_a = argmax(ANN.forward(image))
    samp = N(0, u_var*I, n_real)
    image_as = argmax(ANN.forward(image + samp))   ▷ Expand search window
    while Sum(image_as == y_a) > n_real*tol/2 do
        u_var = u_var*2
        samp = N(0, u_var*I, n_real)
        image_as = argmax(ANN.forward(image + samp))
    end while                                                ▷ perform the bracketing
    for i in range(0,n) do
        count+=1                                 ▷ compute sample and its torch tensor
        samp = N(0, a_var*I, n_real)
        image_as = argmax(ANN.forward(image + samp))
        a_counts[i] = Sum(image_as == y_a)
                                                 ▷ floor and ceiling surround number
        if ((a_counts[i] ≤ Ceil(n_real*(tol*u_tol))) & (a_counts[i] > Floor(n_real*(tol*l_tol)))) then
     return a_var
        else if (a_counts[i] < n_real*tol) then                ▷ we're too high
            u_var = a_var
            a_var = (a_var + l_var)/2
        else if (a_counts[i] ≥ n_real*tol) then                ▷ we're too low
            l_var = a_var
            a_var = (u_var + a_var)/2
        end if
    end for
     return a_var
end function
```

TABLE B.1: Structure of the CNNs C-Ch used in Table 3.1

Layer	Type	Channels	Kernel	Stride	Output Shape
0	Image	1	NA	NA	$(1, 28, 28)$
1	Conv	Ch	$(5, 5)$	$(1, 1)$	$(\text{Ch}, 24, 24)$
2	Conv	Ch	$(5, 5)$	$(1, 1)$	$(\text{Ch}, 20, 20)$
3	Conv	Ch	$(5, 5)$	$(1, 1)$	$(\text{Ch}, 16, 16)$
4	Conv	Ch	$(5, 5)$	$(1, 1)$	$(\text{Ch}, 12, 12)$
5	Max Pool	Ch	$(2, 2)$	$(2, 2)$	$(\text{Ch}, 6, 6)$
7	FC	$(\text{Ch} \cdot 6 \cdot 6, 256)$	NA	NA	256
8	FC	$(256, 10)$	NA	NA	10

B.4 Additional Figures

In this section we provide additional figures to demonstrate some of the experiments from the paper.

B.4.1 Additional figures from MNIST

In Figure B.1 we begin with an image of a `1` and generate adversarial examples to the networks described in Section 3.4.1 via IGSM targeted at each class `2` through `9`; plotted are the counts of output classifications by the DNN from samples from Gaussian distributions with increasing standard deviation; this complements Figure 3.2 in the main text. Note that the prevalence of the adversarial class falls off quickly in all cases, though the rate is different for different choices of target class.

We also show histograms corresponding to those in Figure 3.3 and the networks from Table 3.1. As before, for each image, we used IGSM to generate 9 adversarial examples (one for each target class) yielding a total of 1800 adversarial examples. In addition, we randomly sampled 1800 natural MNIST images. For each of the 3600 images, we computed 0.7-persistence. In Figure B.2, we see histograms of

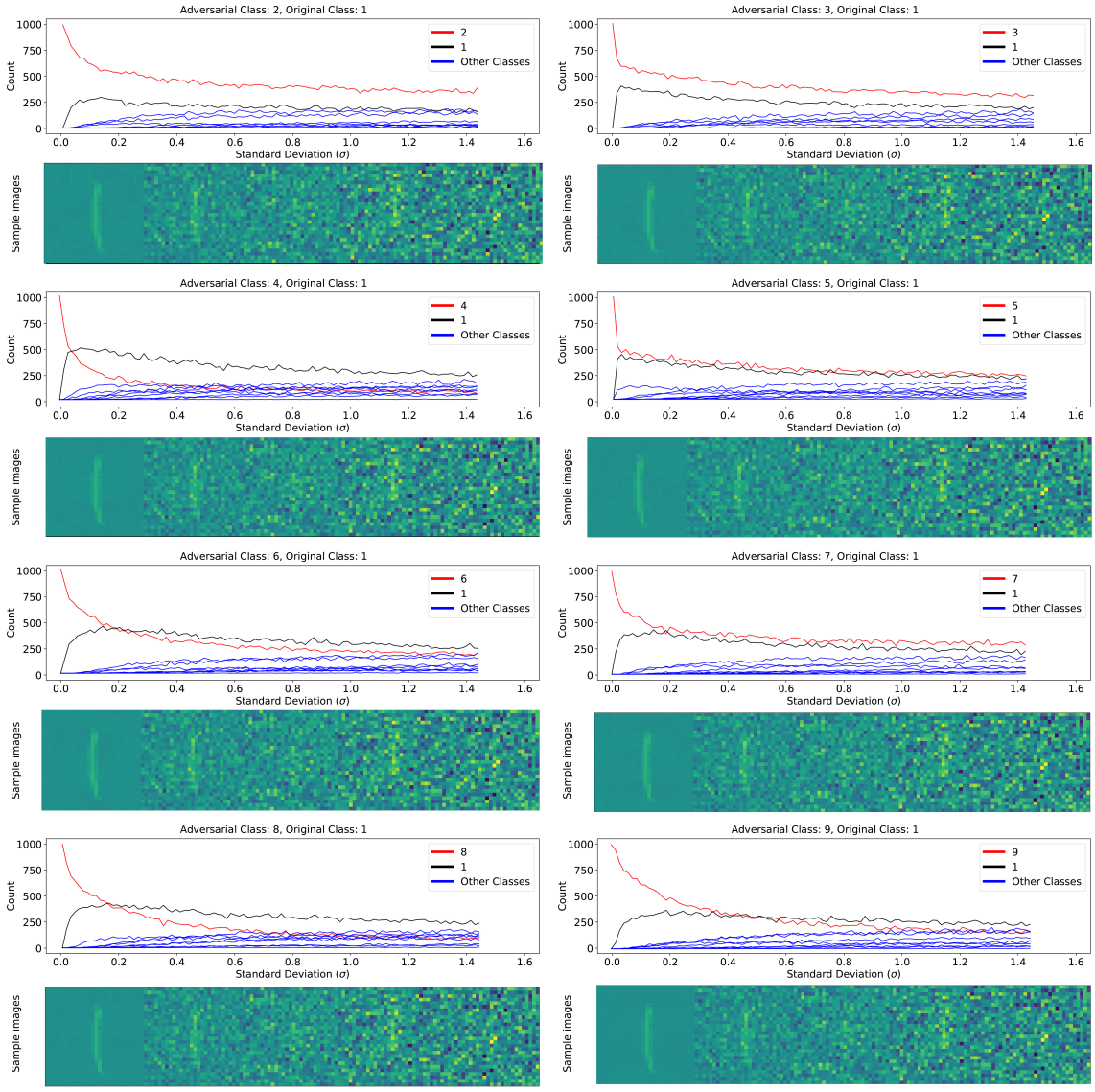


FIGURE B.1: Frequency of each class in Gaussian samples with increasing standard deviations around adversarial attacks of an image of a `1` targeted at classes `2` through `9` on a DNN classifier generated using IGSM. The adversarial class is shown as a red curve. The natural image class (`1`) is shown in black. Bottoms show example sample images at different standard deviations.

these persistences for the small fully connected networks with increasing levels of regularization. In each case, the test accuracy is relatively low and distortion relatively

high. It should be noted that these high-distortion attacks against models with few effective parameters were inherently very stable – resulting in most of the "adversarial" images in these sets having higher persistence than natural images. This suggests a lack of the sharp conical regions which appear to characterize adversarial examples generated against more complicated models. In Figure B.3 we see the larger fully connected networks from Table 3.1 and in Figure B.4 we see some of the convolutional neural networks from Table 3.1.

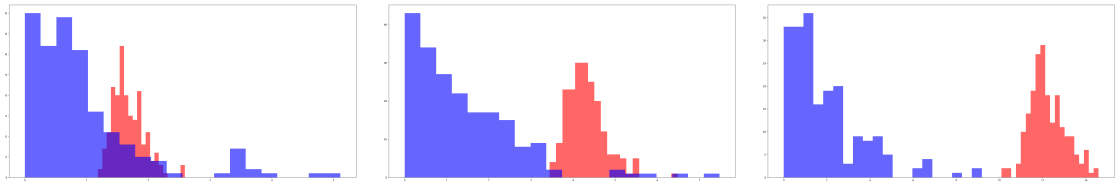


FIGURE B.2: Histograms of 0.7-persistence for FC10-4 (smallest regularization, left), FC10-2 (middle), and FC10-0 (most regularization, right) from Table 3.1. Natural images are in blue, and adversarial images are in red. Note that these are plotted on different scales – higher regularization forces any "adversaries" to be very stable.

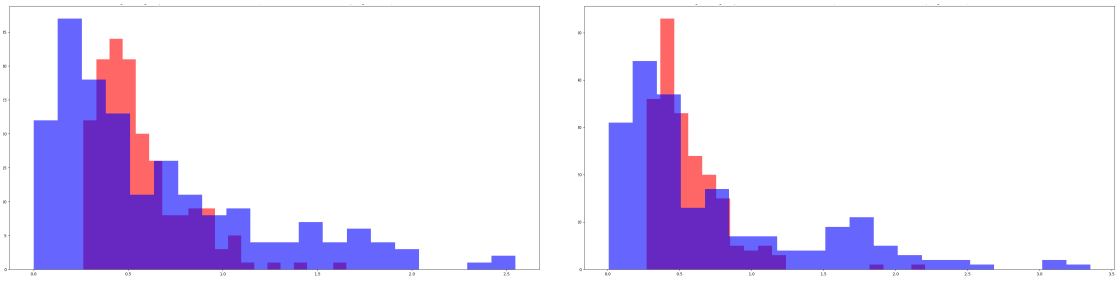


FIGURE B.3: Histograms of 0.7-persistence for FC100-100-10 (left) and FC200-200-10 (right) from Table 3.1. Natural images are in blue, and adversarial images are in red.

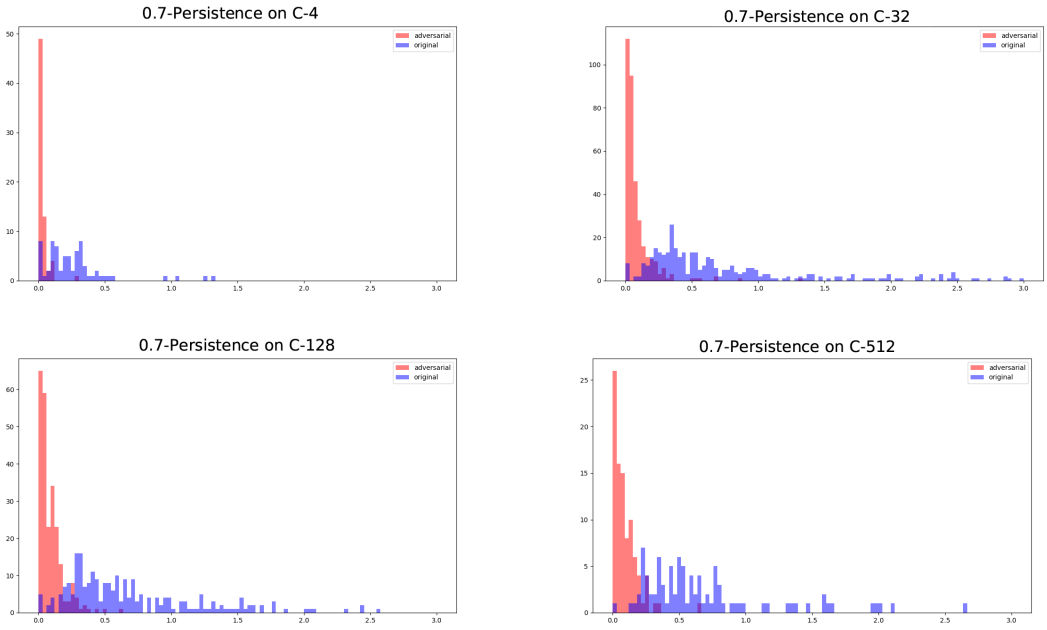


FIGURE B.4: Histograms of 0.7-persistence for C-4 (top left), C-32 (top right), C-128 (bottom left), and C-512 (bottom right) from Table 3.1. Natural images are in blue and adversarial images are in red.

B.4.2 Additional figures for ImageNet

In this section we show some additional figures of Gaussian sampling for ImageNet. In Figure B.5 we see Gaussian sampling of an example of the class `indigo_bunting` and the frequency samplings for adversarial attacks of `goldfinch` toward `indigo_bunting` (classifier: alexnet, attack: PGD) and toward `alligator_lizard` (classifier: vgg16, attack: PGD). Compare the middle image to Figure 3.4, which is a similar adversarial attack but used the vgg16 network classifier and the BIM attack. Results are similar. Also note that in each of the cases in Figure B.5 the label of the original natural image never becomes the most frequent classification when sampling neighborhoods of the adversarial example.

In Figure B.6, we have plotted γ-persistence along a straight line from a natural

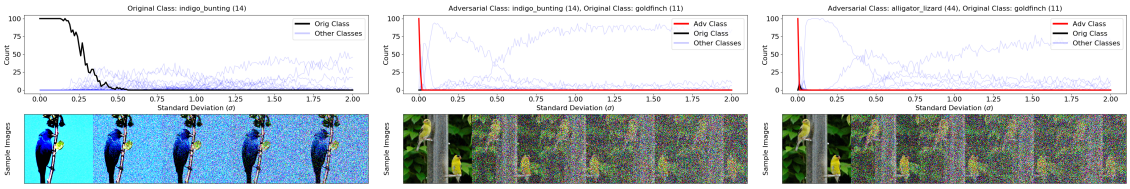


FIGURE B.5: Frequency of each class in Gaussian samples with increasing variance around an `indigo_bunting` image (left), an adversarial example of the image in class `goldfinch` from Figure 3.4 targeted at the `indigo_bunting` class on a alexnet network attacked with PGD (middle), and an adversarial example of the `goldfinch` image targeted at the `alligator_lizard` class on a vgg16 network attacked with PGD (right). Bottoms show example sample images at different standard deviations.

image to an adversarial image to it with differing values of the parameter γ. The γ-persistence in each case seems to change primarily when crossing the decision boundary. Interestingly, while the choice of γ does not make too much of a difference in the left subplot, it leads to more varying persistence values in the right subplot of Figure B.6. This suggests that one should be careful not to choose too small of a γ value, and that persistence does indeed depend on the landscape of the decision boundary described by the classifier.

Algorithm 1 Bracketing algorithm for computing γ-persistence

function BRACKETING(image, classifier ($\mathcal{C}$), numSamples, γ, maxSteps, precision)
  $[\sigma_{\min}, \sigma_{\max}]$ =RANGEFINDER(image, $\mathcal{C}$, numSamples, γ)
  count = 1
  while count<maxSteps **do**
    $\sigma = \frac{\sigma_{\min}+\sigma_{\max}}{2}$
    γ_{new} = COMPUTE_PERSISTENCE(σ, image, numSamples, $\mathcal{C}$)
    if $|\gamma_{\text{new}} - \gamma|$ <precision **then**
      return σ
    else if $\gamma_{\text{new}} > \gamma$ **then**
      $\sigma_{\min} = \sigma$
    else
      $\sigma_{\max} = \sigma$
    end if
    count = count + 1
  end while
  return σ
end function

function RANGEFINDER(image, $\mathcal{C}$, numSamples, γ)
  $\sigma_{\min} = .5, \quad \sigma_{\max} = 1.5$
  γ_1 = COMPUTE_PERSISTENCE($\sigma_{\min}$, image, numSamples, $\mathcal{C}$)
  γ_2 = COMPUTE_PERSISTENCE($\sigma_{\max}$, image, numSamples, $\mathcal{C}$)
  while $\gamma_1 < \gamma$ **or** $\gamma_2 > \gamma$ **do**
    if $\gamma_1 < \gamma$ **then**
      $\sigma_{\min} = .5\sigma_{\min}$
      γ_1 = COMPUTE_PERSISTENCE($\sigma_{\min}$, image, numSamples, $\mathcal{C}$)
    end if
    if $\gamma_2 > \gamma$ **then**
      $\sigma_{\max} = 2\sigma_{\max}$
      γ_2 = COMPUTE_PERSISTENCE($\sigma_{\max}$, image, numSamples, $\mathcal{C}$)
    end if
  end while
  return $[\sigma_{\min}, \sigma_{\max}]$
end function

function COMPUTE_PERSISTENCE(σ, image, numSamples, $\mathcal{C}$)
  sample = N(image, $\sigma^2 I$,numSamples)
  $\gamma_{\text{est}} = \frac{|\{\mathcal{C}(\text{sample})=\mathcal{C}(\text{image})\}|}{\text{numSamples}}$
  return γ_{est}
end function

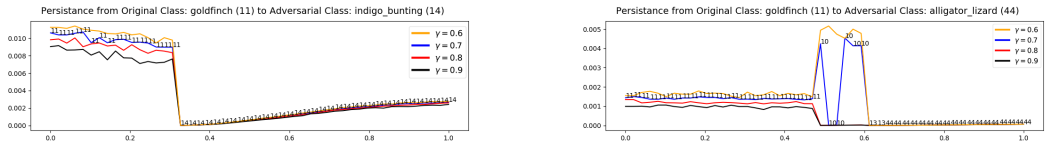


FIGURE B.6: The γ-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) (left) and to an adversarial image generated with PGL in the class `alligator_lizard` (44) (right) on a vgg16 classifier with different values of γ. The classification of each image on the straight line is listed as a number so that it is possible to see the transition from one class to another. The vertical axis is γ-persistence and the horizontal axis is progress towards the adversarial image.

B.5 Concentration of measures

We use Gaussian sampling with varying standard deviation instead of sampling the uniform distributions of balls of varying radius, denoted $U(B_r(0))$ for radius r and center 0. This is for two reasons. The first is that Gaussian sampling is relatively easy to do. The second is that the concentration phenomenon is different. This can be seen in the following proposition.

Proposition B.5.1. *Suppose $x \sim N(0, \sigma^2 I)$ and $y \sim U(B_r(0))$ where both points come from distributions on $\mathbb{R}^n$. For $\varepsilon < \sqrt{n}$ and for $\delta < r$ we find the following:*

$$\mathbb{P}\left[\left|\, \|x\| - \sigma\sqrt{n} \right| \leq \varepsilon\right] \geq 1 - 2e^{-\varepsilon^2/16} \tag{B.1}$$

$$\mathbb{P}\left[\left|\, \|y\| - r \right| \leq \delta\right] \geq 1 - e^{-\delta n/r} \tag{B.2}$$

Proof. This follows from Wegner, 2021, Theorems 4.7 and 3.7, which are the Gaussian Annulus Theorem and the concentration of measure for the unit ball, when taking account of varying the standard deviation σ and radius r, respectively. □

The implication is that if we fix the dimension and let σ vary, the measures will always be concentrated near spheres of radius $\sigma\sqrt{n}$ and r, respectively, in a consistent way. In practice, Gaussians seem to have a bit more spread, as indicated in Figure B.7, which shows the norms of $100,000$ points sampled from dimension $n = 784$ (left, the dimension of MNIST) and $5,000$ points sampled from dimension $n = 196,608$ (right, the dimension of ImageNet).

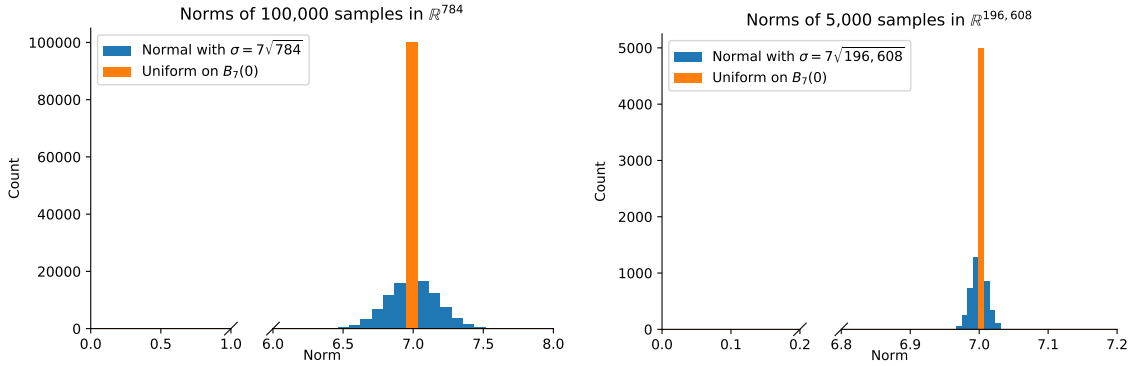


FIGURE B.7: Comparison of the length of samples drawn from $U(B_7(0))$ and $N(0, 7\sqrt{n})$ for $n = 784$, the dimension of MNIST, (left) and $n = 196,608$, the dimension of ImageNet, (right).

Appendix C The EPK is a Kernel

C.0.1 When is an Ensemble of Kernel Machines itself a Kernel Machine?

Here we investigate when our derived ensemble of kernel machines composes to a single kernel machine. In order to show that a linear combination of kernels also equates to a kernel it is sufficient to show that $a_{i,s} = a_{i,0}$ for all $a_{i,s}$. The a_i terms in our kernel machine are determined by the gradient the training loss function. This statement then implies that the gradient of the loss term must be constant throughout training in order to form a kernel. Here we show that this is the case when we consider a log softmax activation on the final layer and a negative log likelihood loss function.

Proof. Assume a two class problem. In the case of a function with multiple outputs, we consider each output to be a kernel. We define our network output $\hat{y}_i$ as all layers up to and including the log softmax and y_i is a one-hot encoded vector.

$$L(\hat{y}_i, y_i) = \sum_{k=1}^{K} -y_i^k(\hat{y}_i^k) \tag{C.1}$$

(C.2)

For a given output indexed by k, if $y_i^k = 1$ then we have

$$L(\hat{y}_i, y_i) = -1(\hat{y}_i^k) \tag{C.3}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = -1 \tag{C.4}$$

$$\tag{C.5}$$

If $y_i^k = 0$ then we have

$$L(\hat{y}_i, y_i) = 0(\hat{y}_i^k) \tag{C.6}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = 0 \tag{C.7}$$

$$\tag{C.8}$$

In this case, since the loss is scaled directly by the output, and the only other term is an indicator function deciding which class label to take, we get a constant gradient. This shows that the gradient of the loss function does not depend on $\hat{y}_i$. Therefore:

$$y = b - \varepsilon \sum_{i=1}^{N} \sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.9}$$

$$= b - \varepsilon \sum_{i=1}^{N} a_{i,0} \sum_{s=1}^{S} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.10}$$

This formulates a kernel machine where

$$a_{i,0} = \frac{\partial L(f_{w_0}(x_i), y_i)}{\partial f_i} \tag{C.11}$$

$$K(x, x_i) = \sum_{s=1}^{S} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.12}$$

$$\phi_{s,t}(x) = \nabla_w f_{w_s(t,x)}(x) \tag{C.13}$$

$$w_s(t, x) = \begin{cases} w_s, x \in X_T \\ w_s(t), x \notin X_T \end{cases} \tag{C.14}$$

$$b = 0 \tag{C.15}$$

□

It is important to note that this does not hold up if we consider the log softmax function to be part of the loss instead of the network. In addition, there are loss structures which can not be rearranged to allow this property. In the simple case of linear regression, we can not disentangle the loss gradients from the kernel formulation, preventing the construction of a valid kernel. For example assume our loss is instead squared error. Our labels are continuous on $\mathbb{R}$ and our activation is the identity function.

$$L(f_i, y_i) = (y_i - f_{i,s})^2 \tag{C.16}$$

$$\frac{\partial L(f_i, y_i)}{\partial f_i} = 2(y_i - f_{i,s}) \tag{C.17}$$

This quantity is dependent on f_i and its value is changing throughout training.

In order for

$$\sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{C.18}$$

to be a kernel on its own, we need it to be a positive (or negative) definite operator and symetric. In the specific case of our practical path kernel, i.e. that in $K(x, x')$ if x' happens to be equal to x_i, then positive semi-definiteness can be accounted for:

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{C.19}$$

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt \tag{C.20}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - f_{i,s} \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt \right) \tag{C.21}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w f_{w_s(t))}(x), f_{i,s} \nabla_w f_{w_s(0)}(x_i) \rangle dt \right) \tag{C.22}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \frac{1}{2} \nabla_w (f_{w_s(0)}(x_i))^2 \rangle dt \right) \tag{C.23}$$

$$\tag{C.24}$$

Otherwise, we get the usual

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \nabla_w f_{w_s(t,x))}(x), \nabla_w f_{w_s(t,x)}(x') \rangle dt \tag{C.25}$$

$$\tag{C.26}$$

The question is two fold. One, in general theory (i.e. the lower example), can we contrive two pairs (x_1, x_1') and (x_2, x_2') that don't necessarily need to be training or test images for which this sum is positive for 1 and negative for 2. Second, in the case that we are always comparing against training images, do we get something more predictable since there is greater dependence on x_i and we get the above way of re-writing using the gradient of the square of $f(x_i)$.

However, even accounting for this by removing the sign of the loss will still produce a non-symmetric function. This limitation is more difficult to overcome.

C.0.2 Multi-Class Case

There are two ways of treating our loss function L for a number of classes (or number of output activations) K:

$$\text{Case 1: } L : \mathbb{R}^K \to \mathbb{R} \tag{C.27}$$

$$\text{Case 2: } L : \mathbb{R}^K \to \mathbb{R}^K \tag{C.28}$$

$$\tag{C.29}$$

Case 1 Scalar Loss

Let $L : \mathbb{R}^K \to \mathbb{R}$. We use the chain rule $D(g \circ f)(x) = Dg(f(x))Df(x)$.

Let f be a vector valued function so that $f : \mathbb{R}^D \to \mathbb{R}^K$ satisfying the conditions from [representation theorem above] with $x \in \mathbb{R}^D$ and $y_i \in \mathbb{R}^K$ for every i. We note that $\dfrac{\partial f}{\partial t}$ is a column and has shape $Kx1$ and our first chain rule can be done the old

fashioned way on each row of that column:

$$\frac{df}{dt} = \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \frac{dw_j}{dt} \tag{C.30}$$

$$= -\varepsilon \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(f(x_i), y_i)}{\partial w_j} \tag{C.31}$$

Apply chain rule (C.32)

$$= -\varepsilon \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(f(x_i), y_i)}{\partial f} \frac{df(x_i)}{\partial w_j} \tag{C.33}$$

Let (C.34)

$$A = \frac{df(x)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{C.35}$$

$$B = \frac{dL(f(x_i), y_i)}{df} \in \mathbb{R}^{1 \times K} \tag{C.36}$$

$$C = \frac{df(x_i)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{C.37}$$

We have a matrix multiplication ABC and we wish to swap the order so somehow we can pull B out, leaving A and C to compose our product for the representation. Since $BC \in \mathbb{R}$, we have $(BC) = (BC)^T$ and we can write

$$(ABC)^T = (BC)^T A^T = BCA^T \tag{C.38}$$

$$ABC = (BCA^T)^T \tag{C.39}$$

Note: This condition needs to be checked carefully for other formulations so that we can re-order the product as follows:

$$= -\varepsilon \sum_{j=1}^{M} \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \frac{df(x_i)}{\partial w_j} \left(\frac{df(x)}{\partial w_j} \right)^T \right)^T \tag{C.40}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \sum_{j=1}^{M} \frac{df(x_i)}{\partial w_j} \left(\frac{df(x)}{\partial w_j} \right)^T \right)^T \tag{C.41}$$

$$\tag{C.42}$$

Note, now that we are summing over j, so we can write this as an inner product on j with the ∇ operator which in this case is computing the jacobian of f along the dimensions of class (index k) and weight (index j). We can define

$$(\nabla f(x))_{k,j} = \frac{df_k(x)}{\partial w_j} \tag{C.43}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \nabla f(x_i) (\nabla f(x))^T \right)^T \tag{C.44}$$

$$\tag{C.45}$$

We note that the dimensions of each of these matrices in order are $[1, K]$, $[K, M]$, and $[M, K]$ which will yield a matrix of dimension $[1, K]$ i.e. a row vector which we then transpose to get back a column of shape $[K, 1]$. Also, we note that our kernel inner product now has shape $[K, K]$.

C.0.3 Schemes Other than Forward Euler (SGD)

Variable Step Size: Suppose f is being trained using Variable step sizes so that across the training set X:

$$\frac{dw_s(t)}{dt} = -\varepsilon_s \nabla_w L(f_{w_s(0)}(X), y_i) = -\varepsilon \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(X), y_i)}{\partial w_j} \tag{C.46}$$

This additional dependence of ε on s simply forces us to keep ε inside the summation in equation 4.23.

Other Numerical Schemes: Suppose f is being trained using another numerical scheme so that:

$$\frac{dw_s(t)}{dt} = \varepsilon_{s,l} \nabla_w L(f_{w_s(0)}(x_i), y_i) + \varepsilon_{s-1,l} \nabla_w L(f_{w_{s-1}}(x_i), y_i) + \cdots \tag{C.47}$$

$$= \varepsilon_{s,l} \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial w_j} + \varepsilon_{s-1,l} \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_{s-1}(0)}(x_i), y_i)}{\partial w_j} + \cdots \tag{C.48}$$

This additional dependence of ε on s and l simply results in an additional summation in equation 4.23. Since addition commutes through kernels, this allows separation into a separate kernel for each step contribution. Leapfrog and other first order schemes will fit this category.

Higher Order Schemes: Luckily these are intractable for for most machine-learning models because they would require introducing dependence of the kernel on input data or require drastic changes. It is an open but intractable problem to derive kernels corresponding to higher order methods.

C.0.4 Variance Estimation

In order to estimate variance we treat our derived kernel function K as the covariance function for Gaussian process regression. Given training data X and test data X', we can use the Kriging to write the mean prediction and it variance from true $\mu(x)$ as

$$\bar{\mu}(X') = \left[K(X', X)\right] \left[K(X, X)\right]^{-1} \left[Y\right] \tag{C.49}$$

$$\mathrm{Var}(\bar{\mu}(X') - \mu(X')) = \left[K(X', X')\right] - \left[K(X', X)\right] \left[K(X, X)\right]^{-1} \left[K(X, X')\right] \tag{C.50}$$

$$\tag{C.51}$$

Where

$$[K(A, B)] = \begin{bmatrix} K(A_1, B_1) & K(A_1, B_2) & \cdots \\ K(A_2, B_1) & K(A_2, B_2) & \\ \vdots & & \ddots \end{bmatrix} \tag{C.52}$$

To finish our Gaussian estimation, we note that each $K(A_i, B_i)$ will me a k by k matrix where k is the number of classes. We take $\mathrm{tr}(K(A_i, B_i)$ to determine total variance for each prediction.

Acknowledgements

This material is based upon work supported by the Department of Energy (National Nuclear Security Administration Minority Serving Institution Partnership Program's

CONNECT - the COnsortium on Nuclear sECurity Technologies) DE-NA0004107. This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

This material is based upon work supported by the National Science Foundation under Grant No. 2134237. We would like to additionally acknowledge funding from NSF TRIPODS Award Number 1740858 and NSF RTG Applied Mathematics and Statistics for Data-Driven Discovery Award Number 1937229. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Bibliography

Abdar, Moloud et al. (2021). "A review of uncertainty quantification in deep learning: Techniques, applications and challenges". In: *Information Fusion* 76, pp. 243–297. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2021.05.008. URL: https://www.sciencedirect.com/science/article/pii/S1566253521001081.

Adebayo, Julius et al. (2018). "Sanity checks for saliency maps". In: *Advances in neural information processing systems* 31.

Alemohammad, Sina et al. (June 2021). *The Recurrent Neural Tangent Kernel*. arXiv:2006.10246 [cs, stat]. DOI: 10.48550/arXiv.2006.10246. URL: http://arxiv.org/abs/2006.10246 (visited on 05/16/2023).

Altekrüger, Fabian, Johannes Hertrich, and Gabriele Steidl (June 2, 2023). *Neural Wasserstein Gradient Flows for Maximum Mean Discrepancies with Riesz Kernels*. arXiv: 2301.11624[cs,math]. URL: http://arxiv.org/abs/2301.11624 (visited on 08/02/2023).

Ansuini, Alessio et al. (2019). "Intrinsic dimension of data representations in deep neural networks". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2019/hash/cfcce0621b49c983991ead4c3d4d3b6b-Abstract.html.

Aparne, Gupta, Andrzej Banburski, and Tomaso Poggio (2022). *PCA as a defense against some adversaries*. Tech. rep. Center for Brains, Minds and Machines (CBMM).

Behpour, Sima et al. (2023). "GradOrth: A Simple yet Efficient Out-of-Distribution Detection with Orthogonal Projection of Gradients". In: *CoRR* abs/2308.00310. DOI: 10.48550/arXiv.2308.00310. arXiv: 2308.00310. URL: https://doi.org/10.48550/arXiv.2308.00310.

Bell, Brian et al. (2023). "An Exact Kernel Equivalence for Finite Classification Models". In: *CoRR* abs/2308.00824. DOI: 10.48550/arXiv.2308.00824. arXiv: 2308.00824. URL: https://doi.org/10.48550/arXiv.2308.00824.

Bengio, Y et al. (2007). *Greedy layer-wise training of deep networks. NIPS 19 (pp. 153–160).*

Bengio, Yoshua, Olivier Delalleau, and Nicolas Roux (2005). "The Curse of Highly Variable Functions for Local Kernel Machines". In: *Advances in Neural Information Processing Systems*. Ed. by Y. Weiss, B. Schölkopf, and J. Platt. Vol. 18. MIT Press. URL: https://proceedings.neurips.cc/paper/2005/file/663772ea088360f95bac3dc7ffb841be-Paper.pdf.

Bengio, Yoshua et al. (2009). "Learning deep architectures for AI". In: *Foundations and trends® in Machine Learning* 2.1, pp. 1–127.

Bietti, Alberto and Julien Mairal (2019). "On the Inductive Bias of Neural Tangent Kernels". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c4ef9c39b300931b69a36fb3dbb8d60e-Paper.pdf.

Biggio, Battista et al. (2014). "Security Evaluation of Support Vector Machines in Adversarial Environments". In: *CoRR* abs/1401.7727. arXiv: 1401.7727. URL: http://arxiv.org/abs/1401.7727.

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc. ISBN: 0387310738.

Blau, Tsachi et al. (2023). *Classifier Robustness Enhancement Via Test-Time Transformation*. arXiv: 2303.15409 [cs.CV].

Blum, Avrim et al. (2020). "Random smoothing might be unable to certify L^∞ robustness for high-dimensional images". In: *The Journal of Machine Learning Research* 21.1, pp. 8726–8746.

Boureau, Y-Lan et al. (2010). "Learning mid-level features for recognition". In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE, pp. 2559–2566.

Bubeck, Sébastien and Mark Sellke (2021). "A Universal Law of Robustness via Isoperimetry". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al., pp. 28811–28822. URL: https://proceedings.neurips.cc/paper/2021/hash/f197002b9a0853eca5e046d9ca4663d5-Abstract.html.

Carlini, Nicholas and David Wagner (Aug. 2016). "Towards Evaluating the Robustness of Neural Networks". In: *arXiv:1608.04644 [cs]*. arXiv: 1608.04644, pp. 39–57. URL: http://arxiv.org/abs/1608.04644 (visited on 04/25/2018).

Ceruti, Claudio et al. (2012). "DANCo: Dimensionality from Angle and Norm Concentration". In: *CoRR* abs/1206.3881. arXiv: 1206.3881. URL: http://arxiv.org/abs/1206.3881.

Cha, Jaehoon and Jeyan Thiyagalingam (July 3, 2023). "Orthogonality-Enforced Latent Space in Autoencoders: An Approach to Learning Disentangled Representations". In: *Proceedings of the 40th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 3913–3948. URL: https://proceedings.mlr.press/v202/cha23b.html (visited on 08/02/2023).

Chen, Chen et al. (2023). "Decision Boundary-Aware Data Augmentation for Adversarial Training". In: *IEEE Transactions on Dependable and Secure Computing* 20.3, pp. 1882–1894. DOI: 10.1109/TDSC.2022.3165889.

Chen, Jiefeng et al. (2021a). "ATOM: Robustifying Out-of-Distribution Detection Using Outlier Mining". In: *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part III*. Ed. by Nuria Oliver et al. Vol. 12977. Lecture Notes in Computer Science. Springer, pp. 430–445. DOI: 10.1007/978-3-030-86523-8_26. URL: https://doi.org/10.1007/978-3-030-86523-8_26.

Chen, Yilan et al. (2021b). "On the equivalence between neural network and support vector machine". In: *Advances in Neural Information Processing Systems* 34, pp. 23478–23490.

Chen, Zixiang et al. (2020). "Mean-Field Analysis of Two-Layer Neural Networks: Non-Asymptotic Rates and Generalization Bounds". In: *CoRR* abs/2002.04026. arXiv: 2002.04026. URL: https://arxiv.org/abs/2002.04026.

Chizat, Lénaïc and Francis Bach (2020). "Implicit Bias of Gradient Descent for Wide Two-layer Neural Networks Trained with the Logistic Loss". In: *Proceedings of Thirty Third Conference on Learning Theory*. Ed. by Jacob Abernethy and

Shivani Agarwal. Vol. 125. Proceedings of Machine Learning Research. PMLR, pp. 1305–1338. URL: https://proceedings.mlr.press/v125/chizat20a.html.

Coates, Adam, Andrew Ng, and Honglak Lee (2011). "An analysis of single-layer networks in unsupervised feature learning". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 215–223.

Cohen, Jeremy, Elan Rosenfeld, and Zico Kolter (2019). "Certified adversarial robustness via randomized smoothing". In: *international conference on machine learning*. PMLR, pp. 1310–1320.

Collobert, Ronan, Samy Bengio, and Johnny Mariéthoz (2002). "Torch: a modular machine learning software library". In:

Collobert, Ronan et al. (2011). "Natural language processing (almost) from scratch". In: *Journal of machine learning research* 12.ARTICLE, pp. 2493–2537.

Costa, J.A. and A.O. Hero (2004a). "Geodesic entropic graphs for dimension and entropy estimation in manifold learning". In: *IEEE Transactions on Signal Processing* 52.8, 2210–2221. ISSN: 1941-0476. DOI: 10.1109/TSP.2004.831130.

Costa, Jose A. and Alfred O. Hero (2004b). "Learning intrinsic dimension and intrinsic entropy of high-dimensional datasets". In: *2004 12th European Signal Processing Conference*, 369–372.

Crecchi, Francesco, Davide Bacciu, and Battista Biggio (2019). "Detecting Adversarial Examples through Nonlinear Dimensionality Reduction". In: *27th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning - ESANN '19*, pp. 483–488.

De Silva, Ashwin et al. (July 13, 2023). *The Value of Out-of-Distribution Data*. DOI: 10.48550/arXiv.2208.10967. arXiv: 2208.10967[cs,stat]. URL: http://arxiv.org/abs/2208.10967 (visited on 08/03/2023).

Deng, Jia et al. (2009). "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. IEEE, pp. 248–255.

Djurisic, Andrija et al. (2023). "Extremely Simple Activation Shaping for Out-of-Distribution Detection". In: *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. URL: https://openreview.net/pdf?id=ndYXTEL6cZz.

Domingos, Pedro (2020a). "Every model learned by gradient descent is approximately a kernel machine". In: *arXiv preprint arXiv:2012.00152*.

— (2020b). "Every Model Learned by Gradient Descent Is Approximately a Kernel Machine". In: *CoRR* abs/2012.00152. arXiv: 2012.00152. URL: https://arxiv.org/abs/2012.00152.

Dominguez-Olmedo, Ricardo et al. (July 3, 2023). "On Data Manifolds Entailed by Structural Causal Models". In: *Proceedings of the 40th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 8188–8201. URL: https://proceedings.mlr.press/v202/dominguez-olmedo23a.html (visited on 08/03/2023).

Dong, Yinpeng et al. "Boosting Adversarial Attacks With Momentum". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR) 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 9185–9193.

Du, Simon S et al. (2019). "Graph Neural Tangent Kernel: Fusing Graph Neural Networks with Graph Kernels". In: *Advances in Neural Information Processing*

Systems. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/663fd3c5144fd10bd5ca6611a9a5Paper.pdf.

Erhan, Dumitru et al. (2010). "Why does unsupervised pre-training help deep learning?" In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 201–208.

Evtimov, Ivan et al. (2017). "Robust Physical-World Attacks on Machine Learning Models". In: *CoRR* abs/1707.08945. arXiv: 1707.08945. URL: http://arxiv.org/abs/1707.08945.

Facco, Elena et al. (2018). "Estimating the intrinsic dimension of datasets by a minimal neighborhood information". In: *CoRR* abs/1803.06992. arXiv: 1803.06992. URL: http://arxiv.org/abs/1803.06992.

Fawzi, Alhussein et al. (2018). "Empirical Study of the Topology and Geometry of Deep Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Filos, Angelos et al. (2020). "Can Autonomous Vehicles Identify, Recover From, and Adapt to Distribution Shifts?" In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 3145–3153. URL: http://proceedings.mlr.press/v119/filos20a.html.

Frosst, Nicholas, Sara Sabour, and Geoffrey E. Hinton (2018). "DARCCC: Detecting Adversaries by Reconstruction from Class Conditional Capsules". In: *CoRR* abs/1811.06969. arXiv: 1811.06969. URL: http://arxiv.org/abs/1811.06969.

Fumera, Giorgio and Fabio Roli (2002). "Support Vector Machines with Embedded Reject Option". In: *Pattern Recognition with Support Vector Machines, First International Workshop, SVM 2002, Niagara Falls, Canada, August 10, 2002, Proceedings*. Ed. by Seong-Whan Lee and Alessandro Verri. Vol. 2388. Lecture Notes in Computer Science. Springer, pp. 68–82. DOI: 10.1007/3-540-45665-1_6. URL: https://doi.org/10.1007/3-540-45665-1_6.

Ganz, Roy, Bahjat Kawar, and Michael Elad (2022). "Do Perceptually Aligned Gradients Imply Adversarial Robustness?" In: *arXiv preprint arXiv:2207.11378*.

Gao, Irena et al. (June 26, 2023). *Out-of-Domain Robustness via Targeted Augmentations*. DOI: 10.48550/arXiv.2302.11861. arXiv: 2302.11861[cs]. URL: http://arxiv.org/abs/2302.11861 (visited on 08/02/2023).

Geifman, Amnon et al. (2020). "On the Similarity between the Laplace and Neural Tangent Kernels". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., pp. 1451–1461. URL: https://proceedings.neurips.cc/paper/2020/hash/1006ff12c465532f8c574aeaa4461b16-Abstract.html (visited on 05/16/2023).

Ghojogh, Benyamin et al. (2021). *Reproducing Kernel Hilbert Space, Mercer's Theorem, Eigenfunctions, Nyström Method, and Use of Kernels in Machine Learning: Tutorial and Survey*. DOI: 10.48550/ARXIV.2106.08443. URL: https://arxiv.org/abs/2106.08443.

Gillette, Andrew and Eugene Kur (2022). "Data-driven geometric scale detection via Delaunay interpolation". In: *arXiv preprint arXiv:2203.05685*.

Gilmer, Justin et al. (2018a). "Adversarial spheres". In: *arXiv preprint arXiv:1801.02774*.

Gilmer, Justin et al. (2018b). "Adversarial Spheres". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=SkthlLkPf.

Giryes, Raja, Yaniv Plan, and Roman Vershynin (2014). "On the Effective Measure of Dimension in the Analysis Cosparse Model". In: *CoRR* abs/1410.0989. arXiv: 1410.0989. URL: http://arxiv.org/abs/1410.0989.

Glielmo, Aldo et al. (2022). "DADApy: Distance-based analysis of data-manifolds in Python". In: *Patterns* 3.10, p. 100589. DOI: 10.1016/j.patter.2022.100589. URL: https://doi.org/10.1016/j.patter.2022.100589.

Glorot, Xavier and Yoshua Bengio (2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 249–256.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio (2011). "Deep sparse rectifier neural networks". In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, pp. 315–323.

Gong, Sixue, Vishnu Naresh Boddeti, and Anil K. Jain (2019). "On the Intrinsic Dimensionality of Image Representations". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, pp. 3987–3996. DOI: 10.1109/CVPR.2019.00411. URL: http://openaccess.thecvf.com/content_CVPR\

_2019/html/Gong_On_the_Intrinsic_Dimensionality_of_Image_Representations_CVPR_2019_paper.html.

Good, Irving J (1963). "Maximum entropy for hypothesis formulation, especially for multidimensional contingency tables". In: *The Annals of Mathematical Statistics* 34.3, pp. 911–934.

Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy (Dec. 2014). "Explaining and Harnessing Adversarial Examples". In: *arXiv:1412.6572 [cs, stat]*. Ed. by Yoshua Bengio and Yann LeCun. arXiv: 1412.6572. URL: http://arxiv.org/abs/1412.6572 (visited on 04/25/2018).

Goodfellow, Ian J. et al. (2013). *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*. arXiv: 1312.6082 [cs.CV].

Halko, Nathan, Per-Gunnar Martinsson, and Joel A Tropp (2011). "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". In: *SIAM review* 53.2, pp. 217–288.

Härdle, Wolfgang et al. (2004). *Nonparametric and semiparametric models*. Vol. 1. Springer.

Hardt, Moritz, Benjamin Recht, and Yoram Singer (2015). "Train faster, generalize better: Stability of stochastic gradient descent". In: *CoRR* abs/1509.01240. arXiv: 1509.01240. URL: http://arxiv.org/abs/1509.01240.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

He, Warren, Bo Li, and Dawn Song (2018). "Decision Boundary Analysis of Adversarial Examples". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BkpiPMbA-.

Hendrycks, Dan and Thomas G. Dietterich (2019). "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. URL: https://openreview.net/forum?id=HJz6tiCqYm.

Hendrycks, Dan and Kevin Gimpel (2017). "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=Hkg4TI9xl.

Hinton, Geoffrey (2010). "A practical guide to training restricted Boltzmann machines". In: *Momentum* 9.1, p. 926.

Hinton, Geoffrey E, Simon Osindero, and Yee-Whye Teh (2006). "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7, pp. 1527–1554.

Hinton, Geoffrey E and Ruslan R Salakhutdinov (2006). "Reducing the dimensionality of data with neural networks". In: *science* 313.5786, pp. 504–507.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural computation* 9.8, pp. 1735–1780.

Hosseini, Hossein, Sreeram Kannan, and Radha Poovendran (2019). "Are Odds Really Odd? Bypassing Statistical Detection of Adversarial Examples". In: *CoRR* abs/1907.12138. arXiv: 1907.12138. URL: http://arxiv.org/abs/1907.12138.

Hu, Shengyuan et al. (2019). "A New Defense Against Adversarial Images: Turning a Weakness into a Strength". In: *Advances in Neural Information Processing Systems*

32 (NeurIPS 2019) Vancouver, BC, Canada. Ed. by Hanna M. Wallach et al., pp. 1633–1644.

Huang, Rui, Andrew Geng, and Yixuan Li (2021a). "On the Importance of Gradients for Detecting Distributional Shifts in the Wild". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al., pp. 677–689. URL: https://proceedings.neurips.cc/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html.

— (2021b). "On the Importance of Gradients for Detecting Distributional Shifts in the Wild". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al., pp. 677–689. URL: https://proceedings.neurips.cc/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html.

Huang, Rui and Yixuan Li (2021). "MOS: Towards Scaling Out-of-Distribution Detection for Large Semantic Space". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, pp. 8710–8719. DOI: 10.1109/CVPR46437.2021.00860. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Huang_MOS_Towards_Scaling_Out-of-Distribution_Detection_for_Large_Semantic_Space_CVPR_2021_paper.html.

Hutchins, Eric M, Michael J Cloppert, Rohan M Amin, et al. (2011). "Intelligence-driven computer network defense informed by analysis of adversary campaigns

and intrusion kill chains". In: *Leading Issues in Information Warfare & Security Research* 1.1, p. 80.

Igoe, Conor et al. (2022). "How Useful are Gradients for OOD Detection Really?" In: *CoRR* abs/2205.10439. DOI: 10.48550/arXiv.2205.10439. arXiv: 2205.10439. URL: https://doi.org/10.48550/arXiv.2205.10439.

Ilyas, Andrew et al. (2019). "Adversarial examples are not bugs, they are features". In: *Advances in neural information processing systems* 32. Ed. by Hanna M. Wallach et al., pp. 125–136.

Incudini, Massimiliano et al. (2022). *The Quantum Path Kernel: a Generalized Quantum Neural Tangent Kernel for Deep Quantum Machine Learning*. arXiv: 2212.11826 [quant-ph].

Ivakhnenko, Alekse Grigorevich and Valentin Grigorévich Lapa (1965). *Cybernetic predicting devices*. CCM Information Corporation.

Jacot, Arthur, Franck Gabriel, and Clément Hongler (2018). "Neural tangent kernel: Convergence and generalization in neural networks". In: *Advances in neural information processing systems* 31.

Jin, Haibo et al. (2022). "ROBY: Evaluating the adversarial robustness of a deep model by its decision boundaries". In: *Information Sciences* 587, pp. 97–122. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2021.12.021. URL: https://www.sciencedirect.com/science/article/pii/S0020025521012421.

Jo, Jason and Yoshua Bengio (2017). "Measuring the tendency of cnns to learn surface statistical regularities". In: *arXiv preprint arXiv:1711.11561*.

Kak, Subhash (1993). "On training feedforward neural networks". In: *Pramana* 40.1, pp. 35–42.

Kaufman, Ilya and Omri Azencot (July 3, 2023). "Data Representations' Study of Latent Image Manifolds". In: *Proceedings of the 40th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 15928–15945. URL: https://proceedings.mlr.press/v202/kaufman23a.html (visited on 08/02/2023).

Kaur, Simran, Jeremy Cohen, and Zachary C Lipton (2019). "Are perceptually-aligned gradients a general property of robust classifiers?" In: *arXiv preprint arXiv:1910.08640*.

Khoury, Marc and Dylan Hadfield-Menell (2018). "On the Geometry of Adversarial Examples". In: *CoRR* abs/1811.00525. arXiv: 1811.00525. URL: http://arxiv.org/abs/1811.00525.

Kim, Hoki (2020). "Torchattacks : A Pytorch Repository for Adversarial Attacks". In: *CoRR* abs/2010.01950.

Kindermans, Pieter-Jan et al. (2019). "The (Un)reliability of Saliency Methods". In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Ed. by Wojciech Samek et al. Vol. 11700. Lecture Notes in Computer Science. Springer, pp. 267–280. DOI: 10.1007/978-3-030-28954-6_14. URL: https://doi.org/10.1007/978-3-030-28954-6_14.

Krause, Andreas (2020). *Introduction to Machine Learning*.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25.

Kumar, Aounon et al. (2020). "Curse of dimensionality on randomized smoothing for certifiable robustness". In: *International Conference on Machine Learning*. PMLR, pp. 5458–5467.

Kurakin, Alexey, Ian Goodfellow, and Samy Bengio (July 2016). "Adversarial examples in the physical world". In: *arXiv:1607.02533 [cs, stat]*. arXiv: 1607.02533. URL: http://arxiv.org/abs/1607.02533 (visited on 04/25/2018).

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10, p. 1995.

LeCun, Yann and Corinna Cortes (2010). "MNIST handwritten digit database". In: URL: http://yann.lecun.com/exdb/mnist/.

LeCun, Yann et al. (1988). "A theoretical framework for back-propagation". In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann, pp. 21–28.

LeCun, Yann et al. (1989). "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4, pp. 541–551.

LeCun, Yann et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Lecuyer, Mathias et al. (2019). "Certified robustness to adversarial examples with differential privacy". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, pp. 656–672.

Lee, Honglak et al. (2009). "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In: *Proceedings of the 26th annual international conference on machine learning*, pp. 609–616.

Lee, Jaehoon et al. (2018a). "Deep Neural Networks as Gaussian Processes". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=B1EA-M-0Z.

Lee, Jinsol and Ghassan AlRegib (2020). "Gradients as a Measure of Uncertainty in Neural Networks". In: *IEEE International Conference on Image Processing, ICIP 2020, Abu Dhabi, United Arab Emirates, October 25-28, 2020*. IEEE, pp. 2416–2420. DOI: 10.1109/ICIP40778.2020.9190679. URL: https://doi.org/10.1109/ICIP40778.2020.9190679.

Lee, Kimin et al. (2018b). "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *NeurIPS*.

— (2018c). "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *CoRR* abs/1807.03888. arXiv: 1807.03888. URL: http://arxiv.org/abs/1807.03888.

Levina, Elizaveta and Peter Bickel (2004). "Maximum Likelihood Estimation of Intrinsic Dimension". In: *Advances in Neural Information Processing Systems*. Vol. 17. MIT Press. URL: https://proceedings.neurips.cc/paper_files/paper/2004/hash/74934548253bcab8490ebd74afed7031-Abstract.html.

Li, Bai et al. (2019). "Certified adversarial robustness with additive noise". In: *Advances in neural information processing systems* 32.

Li, Yuanzhi and Yang Yuan (2017). "Convergence Analysis of Two-layer Neural Networks with ReLU Activation". In: *Advances in Neural Information Processing Systems*, pp. 597–607.

Liang, Shiyu, Yixuan Li, and R. Srikant (2018). "Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=H1VGkIxRZ.

Lin, Ziqian, Sreya Dutta Roy, and Yixuan Li (2021). "MOOD: Multi-Level Out-of-Distribution Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, pp. 15313–15323. DOI: 10.1109/CVPR46437.2021.01506. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Lin_MOOD_Multi-Level_Out-of-Distribution_Detection_CVPR_2021_paper.html.

Linnainmaa, Seppo (1970). "The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors". In: *Master's Thesis (in Finnish), Univ. Helsinki*, pp. 6–7.

Liu, Dong C and Jorge Nocedal (1989). "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1-3, pp. 503–528.

Liu, Liangchen, Juncai He, and Richard Tsai (July 22, 2023). *Linear Regression on Manifold Structured Data: the Impact of Extrinsic Geometry on Solutions*. DOI: 10.48550/arXiv.2307.02478. arXiv: 2307.02478[cs,math]. URL: http://arxiv.org/abs/2307.02478 (visited on 09/13/2023).

Liu, Shuying and Weihong Deng (2015). "Very deep convolutional neural network based image classification using small training sample size". In: *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*. IEEE, pp. 730–734.

Liu, Weitang et al. (2020). "Energy-based Out-of-distribution Detection". In: *CoRR* abs/2010.03759. arXiv: 2010.03759. URL: https://arxiv.org/abs/2010.03759.

Lu, Yulong and Jianfeng Lu (2020). "A universal approximation theorem of deep neural networks for expressing probability distributions". In: *Advances in neural information processing systems* 33, pp. 3094–3105.

Lu, Zhiping et al. (2022). "MR2D: Multiple Random Masking Reconstruction Adversarial Detector". In: *2022 10th International Conference on Information Systems and Computing Technology (ISCTech)*, pp. 61–67. DOI: 10.1109/ISCTech58360.2022.00016.

Lundberg, Scott and Su-In Lee (2017). *A Unified Approach to Interpreting Model Predictions*. DOI: 10.48550/ARXIV.1705.07874. URL: https://arxiv.org/abs/1705.07874.

Madry, Aleksander et al. (June 2017). "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *arXiv:1706.06083 [cs, stat]*. arXiv: 1706.06083. URL: http://arxiv.org/abs/1706.06083 (visited on 04/25/2018).

— (2018). "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rJzIBfZAb.

Magai, German and Anton Ayzenberg (2022). *Topology and geometry of data manifold in deep learning*. arXiv: 2204.08624 [cs.LG].

Malik, Jitendra and Pietro Perona (1990). "Preattentive texture discrimination with early vision mechanisms". In: *JOSA A* 7.5, pp. 923–932.

Marbut, Anna, Katy Mckinney-Bock, and Travis Wheeler (July 3, 2023). "Reliable Measures of Spread in High Dimensional Latent Spaces". In: *Proceedings of the 40th International Conference on Machine Learning*. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 23871–23885. URL: https://proceedings.mlr.press/v202/marbut23a.html (visited on 08/01/2023).

McClelland, James L, David E Rumelhart, PDP Research Group, et al. (1986). "Parallel distributed processing". In: *Explorations in the Microstructure of Cognition* 2, pp. 216–271.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

Mikolov, Tomas et al. (2010). "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 3. Makuhari, pp. 1045–1048.

Minsky, Marvin and Seymour Papert (1969). "Perceptrons: Anlntroduction to computational geometry". In: *MITPress, Cambridge, Massachusetts*.

Mohseni, Sina et al. (2020). "Self-Supervised Learning for Generalizable Out-of-Distribution Detection". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, pp. 5216–5223. DOI: 10.1609/aaai.v34i04.5966. URL: https://doi.org/10.1609/aaai.v34i04.5966.

Moosavi-Dezfooli, Seyed-Mohsen, Alhussein Fawzi, and Pascal Frossard (Nov. 2015). "DeepFool: a simple and accurate method to fool deep neural networks". In:

arXiv:1511.04599 [cs]. arXiv: 1511.04599. URL: http://arxiv.org/abs/1511.04599 (visited on 04/25/2018).

Nagler, Thomas (2023). "Statistical Foundations of Prior-Data Fitted Networks". In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 25660–25676. URL: https://proceedings.mlr.press/v202/nagler23a.html.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified Linear Units Improve Restricted Boltzmann Machines". en. In: pp. 807–814.

Neal, Radford M and Radford M Neal (1996). "Priors for infinite networks". In: *Bayesian learning for neural networks*, pp. 29–53.

Nguyen Minh, Dang and Anh Tuan Luu (Dec. 2022). "Textual Manifold-based Defense Against Natural Language Adversarial Examples". In: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, pp. 6612–6625. URL: https://aclanthology.org/2022.emnlp-main.443.

Oh, Junsoo and Chulhee Yun (June 5, 2023). *Provable Benefit of Mixup for Finding Optimal Decision Boundaries*. DOI: 10.48550/arXiv.2306.00267. arXiv: 2306.00267[cs,math,stat]. URL: http://arxiv.org/abs/2306.00267 (visited on 08/02/2023).

Osada, Genki et al. (2023). "Out-of-Distribution Detection With Reconstruction Error and Typicality-Based Penalty". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 5551–5563.

Papernot, Nicolas et al. (Nov. 2015). "The Limitations of Deep Learning in Adversarial Settings". In: *arXiv:1511.07528 [cs, stat]*. arXiv: 1511.07528. URL: http://arxiv.org/abs/1511.07528 (visited on 04/25/2018).

Papernot, Nicolas et al. (Oct. 2016). "cleverhans v2.0.0: an adversarial machine learning library". In: *arXiv:1610.00768 [cs, stat]*. arXiv: 1610.00768. URL: http://arxiv.org/abs/1610.00768 (visited on 04/25/2018).

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Petersen, Philipp and Felix Voigtlaender (2018). "Optimal approximation of piecewise smooth functions using deep ReLU neural networks". In: *Neural Networks* 108, pp. 296–330.

Pillai, Ignazio, Giorgio Fumera, and Fabio Roli (2013). "Multi-label classification with a reject option". In: *Pattern Recognit.* 46.8, pp. 2256–2266. DOI: 10.1016/j.patcog.2013.01.035. URL: https://doi.org/10.1016/j.patcog.2013.01.035.

Prakash, Aaditya et al. (2018). "Deflecting Adversarial Attacks with Pixel Deflection". In: *CoRR* abs/1801.08926, pp. 8571–8580. arXiv: 1801.08926. URL: http://arxiv.org/abs/1801.08926.

Qin, Yao et al. "Detecting and Diagnosing Adversarial Images with Class-Conditional Capsule Reconstructions". In: *8th International Conference on Learning Representations, (ICLR 2020), Addis Ababa, Ethiopia*.

Rasmussen, Carl Edward, Christopher KI Williams, et al. (2006). *Gaussian processes for machine learning*. Vol. 1. Springer.

Rosenblatt, Frank (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6, p. 386.

Roth, Kevin, Yannic Kilcher, and Thomas Hofmann (2019). "The Odds are Odd: A Statistical Test for Detecting Adversarial Examples". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. Ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97, pp. 5498–5507.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". In: *nature* 323.6088, pp. 533–536.

Russakovsky, Olga et al. (2015). "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision (IJCV)* 115.3, pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

Schmidhuber, JÃ¼rgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0893608014002135.

Selvaraju, Ramprasaath R. et al. (2019). "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". In: *International Journal of Computer Vision* 128.2, pp. 336–359. DOI: 10.1007/s11263-019-01228-7. URL: https://doi.org/10.1007%2Fs11263-019-01228-7.

Shafahi, Ali et al. (2018). "Are adversarial examples inevitable?" In: *CoRR* abs/1809.02104. arXiv: 1809.02104. URL: http://arxiv.org/abs/1809.02104.

Shah, Harshay, Prateek Jain, and Praneeth Netrapalli (2021). "Do input gradients highlight discriminative features?" In: *Advances in Neural Information Processing Systems* 34, pp. 2046–2059.

Shamir, Adi (2021). "A new theory of adversarial examples in machine learning (a non-technical extended abstract)". In: *preprint*.

Shamir, Adi, Odelia Melamed, and Oriel BenShmuel (2021). "The Dimpled Manifold Model of Adversarial Examples in Machine Learning". In: *CoRR* abs/2106.10151. arXiv: 2106.10151. URL: https://arxiv.org/abs/2106.10151.

Shawe-Taylor, John, Nello Cristianini, et al. (2004). *Kernel methods for pattern analysis*. Cambridge university press.

Shilton, Alistair et al. (Jan. 31, 2023). *Gradient Descent in Neural Networks as Sequential Learning in RKBS*. DOI: 10.48550/arXiv.2302.00205. arXiv: 2302.00205[cs,stat]. URL: http://arxiv.org/abs/2302.00205 (visited on 09/13/2023).

Silva, Ashwin De et al. (2023). "The Value of Out-of-Distribution Data". In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 7366–7389. URL: https://proceedings.mlr.press/v202/de-silva23a.html.

Simchowitz, Max et al. (2023). "Statistical Learning under Heterogenous Distribution Shift". In: *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 31800–31851. URL: https://proceedings.mlr.press/v202/simchowitz23a.html.

Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman (2013). "Deep inside convolutional networks: Visualising image classification models and saliency maps". In: *arXiv preprint arXiv:1312.6034*.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*.

Snelson, Edward and Zoubin Ghahramani (2005). "Sparse Gaussian processes using pseudo-inputs". In: *Advances in neural information processing systems* 18.

Snoek, Jasper et al. (2019). "Can you trust your model's uncertainty? Evaluating predictive uncertainty under dataset shift". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. Ed. by Hanna M. Wallach et al., pp. 13969–13980. URL: https://proceedings.neurips.cc/paper/2019/hash/8558cb408c1d76621371888657d2eb1d-Abstract.html.

Song, Yang et al. (2018). "PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. URL: https://openreview.net/forum?id=rJUYGxbCW.

Srinivas, Suraj, Sebastian Bordt, and Hima Lakkaraju (2023). "Which Models have Perceptually-Aligned Gradients? An Explanation via Off-Manifold Robustness". In: *CoRR* abs/2305.19101. DOI: 10.48550/arXiv.2305.19101. arXiv: 2305.19101. URL: https://doi.org/10.48550/arXiv.2305.19101.

Sun, Yiyou, Chuan Guo, and Yixuan Li (2021). "ReAct: Out-of-distribution Detection With Rectified Activations". In: *Advances in Neural Information Processing Systems*

34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. Ed. by Marc'Aurelio Ranzato et al., pp. 144–157. URL: https://proceedings.neurips.cc/paper/2021/hash/01894d6f048493d2cacde3c579c3Abstract.html.

Sun, Yiyou and Yixuan Li (2022). "DICE: Leveraging Sparsification for Out-of-Distribution Detection". In: *Computer Vision - ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXIV*. Ed. by Shai Avidan et al. Vol. 13684. Lecture Notes in Computer Science. Springer, pp. 691–708. DOI: 10.1007/978-3-031-20053-3_40. URL: https://doi.org/10.1007/978-3-031-20053-3_40.

Sun, Yiyou et al. (2022). "Out-of-distribution Detection with Deep Nearest Neighbors". In: *CoRR* abs/2204.06507. DOI: 10.48550/arXiv.2204.06507. arXiv: 2204.06507. URL: https://doi.org/10.48550/arXiv.2204.06507.

Swaminathan, Sridhar et al. (2020). "Sparse low rank factorization for deep neural network compression". In: *Neurocomputing* 398, pp. 185–196. DOI: 10.1016/j.neucom.2020.02.035. URL: https://doi.org/10.1016/j.neucom.2020.02.035.

Szegedy, Christian et al. (2013). "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199*.

Szegedy, Christian et al. (2014). "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1312.6199.

Talwalkar, Ameet, Sanjiv Kumar, and Henry Rowley (2008). "Large-scale manifold learning". In: *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 1–8. DOI: 10.1109/CVPR.2008.4587670.

Tancik, Matthew et al. (2020). "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 7537–7547. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/55053683268957697aa39fba6f231c68-Paper.pdf.

Taori, Rohan et al. (2020). "Measuring Robustness to Natural Distribution Shifts in Image Classification". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 18583–18599. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/d8330f857a17c53d217014ee776bfd50-Paper.pdf.

Tjeng, Vincent, Kai Xiao, and Russ Tedrake (2017). "Evaluating robustness of neural networks with mixed integer programming". In: *arXiv preprint arXiv:1711.07356*.

Tramer, Florian and Dan Boneh (2019). "Adversarial training and robustness for multiple perturbations". In: *Advances in Neural Information Processing Systems* 32.

Tramèr, Florian et al. "Ensemble Adversarial Training: Attacks and Defenses". In: *6th International Conference on Learning Representations, (ICLR 2018), Vancouver, BC, Canada*.

Tramèr, Florian et al. "On Adaptive Attacks to Adversarial Example Defenses". In: *Advances in Neural Information Processing Systems 33 (NeurIPS 2020), virtual*. Ed. by Hugo Larochelle et al.

Tsipras, Dimitris et al. (2018). "Robustness may be at odds with accuracy". In: *stat* 1050, p. 11.

Vardi, Gal, Gilad Yehudai, and Ohad Shamir (2022). *Gradient Methods Provably Converge to Non-Robust Networks*. arXiv: 2202.04347 [cs.LG].

Vaswani, Ashish et al. (2017). *Attention Is All You Need*. arXiv: 1706.03762 [cs.CL].

Vincent, Pascal et al. (2010). "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." In: *Journal of machine learning research* 11.12.

Wang, Haoran et al. (2021). "Can multi-label classification networks know what they don't know?" In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al., pp. 29074–29087. URL: https://proceedings.neurips.cc/paper/2021/hash/f3b7e5d3eb074cde5b76e26bc0fb Abstract.html.

Wang, Yisen et al. (2020). "Improving Adversarial Robustness Requires Revisiting Misclassified Examples". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rklOg6EFwS.

Wegner, Sven-Ake (2021). "Lecture notes on high-dimensional spaces". In: *arXiv preprint arXiv:2101.05841*.

Werbos, Paul J (1974). "Beyond regression: New tools for prediction and analysis in the be havioral sciences/'PhD diss., Harvard Uni versity. Werbos, Paul J. 1988". In: *Generalization of back propagation with application to a recurrent gas market method," Neural Networks* 1.4, pp. 339–356.

Wiyatno, Rey and Anqi Xu (2018). "Maximal Jacobian-based Saliency Map Attack". In: *CoRR* abs/1808.07945. arXiv: 1808.07945. URL: http://arxiv.org/abs/1808.07945.

Xu, Mingyu et al. (2023a). *VRA: Variational Rectified Activation for Out-of-distribution Detection*. arXiv: 2302.11716 [cs.LG].

Xu, Yuancheng et al. (2023b). *Exploring and Exploiting Decision Boundary Dynamics for Adversarial Robustness*. arXiv: 2302.03015 [cs.LG].

Yang, Greg et al. (2020a). "Randomized smoothing of all shapes and sizes". In: *International Conference on Machine Learning*. PMLR, pp. 10693–10705.

Yang, Huanrui et al. (2020b). "Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. Computer Vision Foundation / IEEE, pp. 2899–2908. DOI: 10.1109/CVPRW50498.2020.00347. URL: https://openaccess.thecvf.com/content_CVPRW_2020/html/w40/Yang_Learning_Low-Rank_Deep_Neural_Networks_via_Singular_Vector_Orthogonality_Regularization_CVPRW_2020_paper.html.

Yang, Jingkang et al. (2021). "Generalized Out-of-Distribution Detection: A Survey". In: *CoRR* abs/2110.11334. arXiv: 2110.11334. URL: https://arxiv.org/abs/2110.11334.

Yang, Yunfei, Zhen Li, and Yang Wang (2022). "On the capacity of deep generative networks for approximating distributions". In: *Neural networks* 145, pp. 144–154.

Yousefzadeh, Roozbeh (2021). "Deep learning generalization and the convex hull of training sets". In: *arXiv preprint arXiv:2101.09849*.

Zhang, Haizhang, Yuesheng Xu, and Jun Zhang (2009). "Reproducing Kernel Banach Spaces for Machine Learning." In: *Journal of Machine Learning Research* 10.12.

Zhao, Shubin and Ralph Grishman (2005). "Extracting relations with integrated information using kernel methods". In: *Proceedings of the 43rd annual meeting of the association for computational linguistics (acl'05)*, pp. 419–426.

Zheng, Yijia et al. (2022). "Learning Manifold Dimensions with Conditional Variational Autoencoders". en. In: *Advances in Neural Information Processing Systems* 35, 34709–34721.

Šípka, Martin et al. (Jan. 27, 2023). *Differentiable Simulations for Enhanced Sampling of Rare Events*. DOI: 10.48550/arXiv.2301.03480. arXiv: 2301.03480[physics]. URL: http://arxiv.org/abs/2301.03480 (visited on 08/02/2023).