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....

"*Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism.*"

Dave Barry

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

Physical Constants

Speed of Light $\quad c_0 = 2.997\,924\,58 \times 10^8\,\mathrm{m\,s^{-1}}$ (exact)

List of Symbols

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

ABSTRACT

The Thesis Abstract is written here (and usually kept to just this page). The page is kept centered vertically so can expand into the blank space above the title too...

Chapter 1 Introduction

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. **goodfellow2013multidigit**. 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 Szegedy et al., 2013 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.

1.0.1 Artificial Neural Networks (ANNs)

The history of Neural Networks 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 **mcculloch1943logical**. 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 by Rosenblatt in 1958 **rosenblatt1958perceptron**. 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 **ivakhnenko1965cybernetic**. 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 in 1969: A proof that basic perceptrons could not encode exclusive-or **minsky1969perceptrons**. 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 **linnainmaa1970representation**. 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**werbos1974beyond** 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 Rumelhard and McClelland in 1986 **mcclelland1986parallel** 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 refined backpropagation into the form used today **lecun1989backpropagation**, invented Convolutional Neural Networks 1.0.2 **lecun1995convolutional**, and by 1998, he had worked with his team to implement what rapidly became the industry standard for banks to recognize hand-written numbers **lecun1998gradient**.

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 from 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. **bengio2007greedy**; **hinton2006reducing**; **hinton2006fast** 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. **coates2011analysis**; **vincent2010stacked**; **boureau2010learning**; **hinton2010practical**; **glorot2010understanding**; **erhan2010does**; **bengio2009learning** With this expansion in theory came a boom in the ability to practically implement more structured and capable neural networks: recurrent networks **mikolov2010recurrent**, convolutional neural networks **lee2009convolutional**, natural language processing **collobert2011natural**, and Long-Short-Term-Memory (LSTM) networks originally developed by Horchreiter (a student of Shmidhuber) in 1997 **hochreiter1997long** 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 inexorable expansion of well-maintained libraries for working with neural networks including the very early creation of the now famous Torch. **Collobert2002TorchAM** and its python interface that still dominates the market-share of machine learning: PyTorch **pytorch2019** 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 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., 2013. Early versions of this network took 1-2 million dollars worth of compute-time to train. 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. **vaswani2017attention**. 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.

1.0.2 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 1.0.1. *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* $b \in \mathbb{R}$*:*

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

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

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 with sigmoids **malik1990preattentive** 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 **glorot2011deep** and additionally training algorithms on ReLU activated networks converge faster **nair_rectified_nodate**.

Definition 1.0.2. *The Rectified Linear Unit (ReLU) function is*

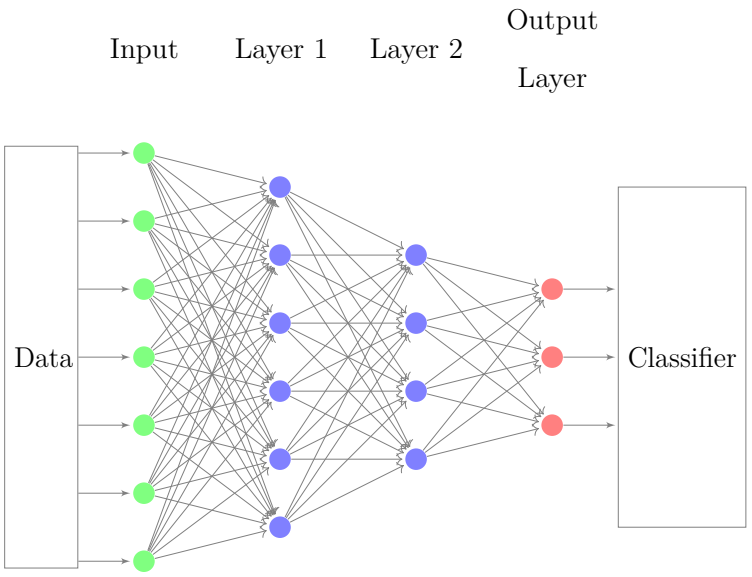
$$ReLU(x) = \begin{cases} 0, & x \leq 0; \\ x, & x > 0, \end{cases}$$

The 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 **petersen2018optimal**. In addition, ReLU is convex, which enables efficient numerical approximation of smooth functions in shallow networks **li2017convergence**.

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 **minsky1969perceptrons** that single layers of perceptrons could not encode exclusive-or. Depth, the number of layers in a neural network, is a key factor in its ability to approximate complicated functions including exclusive-or **kak1993training**.

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 1.0.3. *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 1.0.4. *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:PRM:1162264

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 to be very effective on problems with spatial information **lecun1995convolutional**.

1.0.3 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 **rumelhart1986learning**, dubbed backpropagation. This technique was refined into its modern form in the thesis and continuing work of Yann LeCun **lecun1988theoretical**. 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:PRM:1162264**. 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 **liu2015very**. 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 1.0.5. *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 1.0.6. *Given a loss function L, the Empirical Risk over a training dataset (X, Y) of size N is*

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

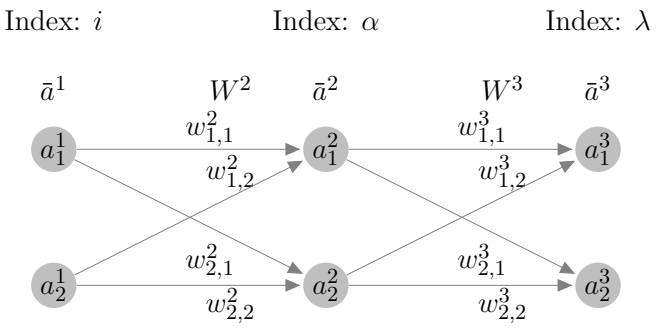
We seek parameters $\vec{w}$ which will minimize $R_{\text{emp}}(P_w(x))$. This will be done with gradient-based optimization.

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 $\bar{A}^n$, we can now write the output $\bar{a}_n$ for any layer of an arbitrary ANN in two ways **Krause20**. Recursively, we can define

$$a_\lambda^n = A^n(\sum_\alpha w_{\alpha,\lambda}^n a_\alpha^{n-1}) \tag{1.1}$$

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 $\dfrac{\partial L}{\partial_{w_{i,j}^l}}$ 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:

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

Many of the terms of this gradient (e.g. the activations a^n_i and the sums $\sum_i w^n_{i,j}a_i$) 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 $\frac{\partial L}{\partial w^l_{n,i}} = \delta^l_n a^l_i$ where δ^l_n will contain terms which are either pre-computed or can be computed analytically. Conveniently, we can define this error signal recursively:

$$\delta^l_n = A'^l(a^l_n)\sum_{i=1}^{n} w^{l+1}_{i,n}\delta^{l+1}_i$$

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, poor guesses for such a large number of parameters can lead to gradients blowing up or down **Bishop:2006:PRM:1162264**. . Due to nonlinearity 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 1.0.7. *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 **HardtRS15**. It is for this reason that SGD has been adopted as the de facto standard among ANN training applications.

Chapter 2 Adversarial Attacks

2.1 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., 2013. Adversarial examples occur when natural data are close enough to the decision boundary that they can be imperceptibly perturbed to another class. Adversarial examples are not just a peculiarity, but seem to occur for most, if not all, DNN classifiers. For example, **shafahi2018are** 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. 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. These were found for a variety of DNNs 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.

In general *Adversarial Attacks* against models are small perturbations from natural data which significantly perturb model output. Szegedy et al. Szegedy et al., 2013 realized that the same computational tools used to train ANNs could be used to generate attacks that would confuse them. 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:

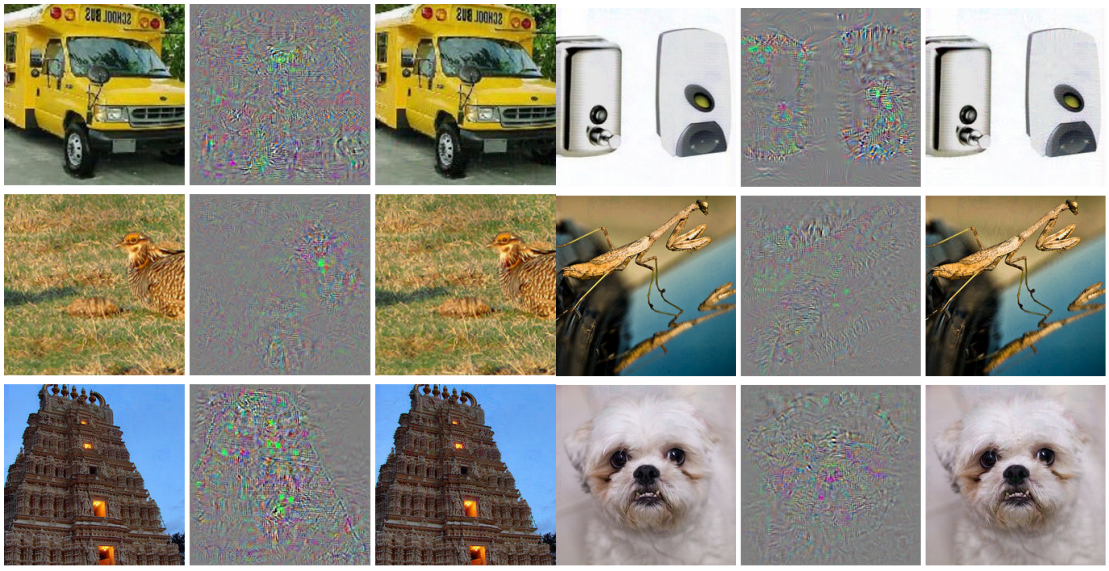


FIGURE 2.1: Natural Images are in columns 1 and 4, Adversarial images are in columns 3 and 6, and the difference between them (magnified by a factor of 10) is in columns 2 and 5. All images in columns 3 and 6 are classified by AlexNet as "Ostrich" Szegedy et al., 2013

2.2 Common Datasets

The Dataset used above is known as ImageNet – a large set of labeled images varying in size originally compiled for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This dataset and its many subsets 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 (MNIST) dataset 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 is therefore experiments run much more quickly on it and require less complex input/output.

2.3 Attack Techniques

2.3.1 L-BFGS minimizing distortion

Szegedy et al. took advantage of the tools they had on hand for training neural networks to set up a box-constrained optimization problem whose approximated solution generates these targeted misclassifications.

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 .
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.

L-BFGS: Mnist The following examples are prepared by implementing the above technique via pytorch on images from the Mnist dataset with FC200-200-10, a neural network with 2 hidden layers with 200 nodes each:

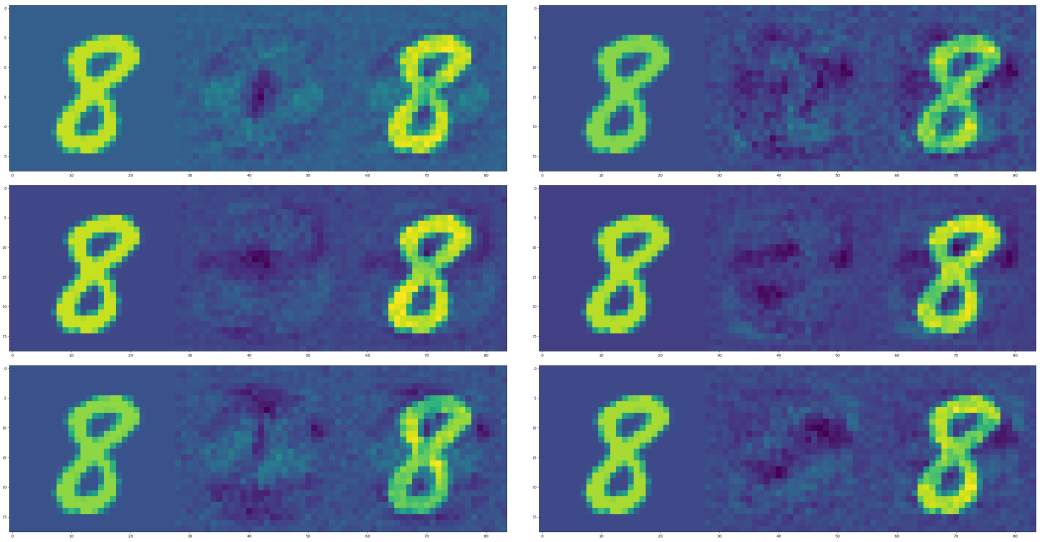


FIGURE 2.2: Original images on the left, Perturbation is in the middle, Adversarial Image (total of Original with Perturbation) is on the right. Column 1 shows an original 8 being perturbed to adversarial classes 0, 2, and 4. Column 2 shows adversarial classes 1, 3, and 5

Borrowing a metric from Szegedy et al to compare the magnitude of these distortions, we will define

Definition 2.3.1. *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 dimensions.

The 900 examples generated for the network above had an average distortion of 0.089 with the following distribution of distortions, given in figure 3.

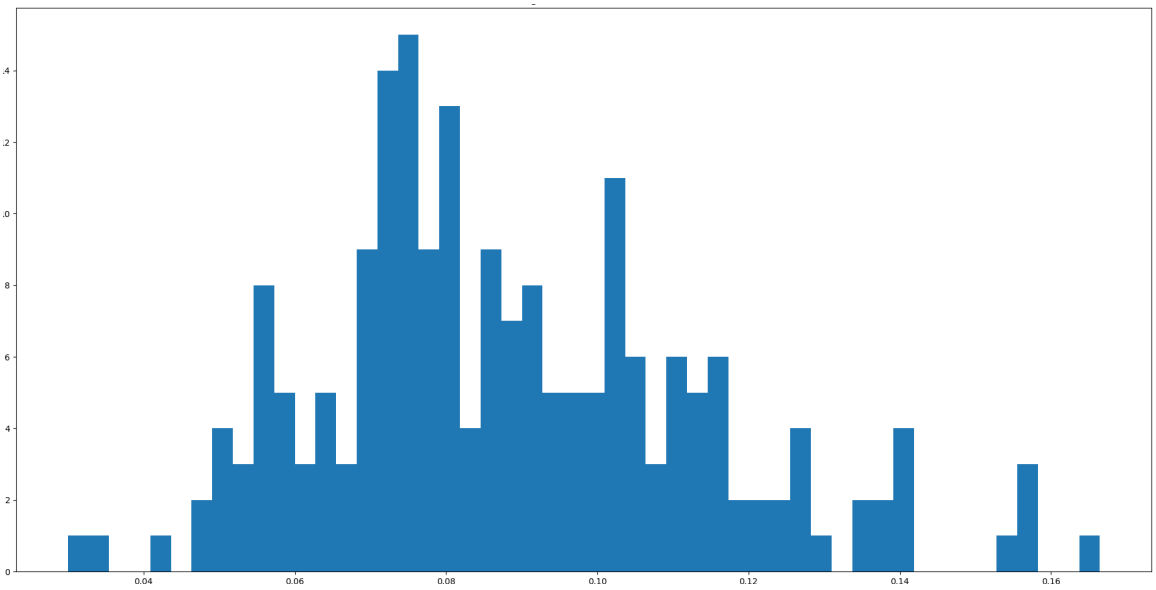


FIGURE 2.3: A histogram of the distortion measured for each of 900 adversarial examples generated using L-BFGS against the FC-200-200-10 network on Mnist. Mean distortion is 0.089.

L-BFGS: ImageNet We also tried to replicate Szegedy et al., 2013's results 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 4:

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.

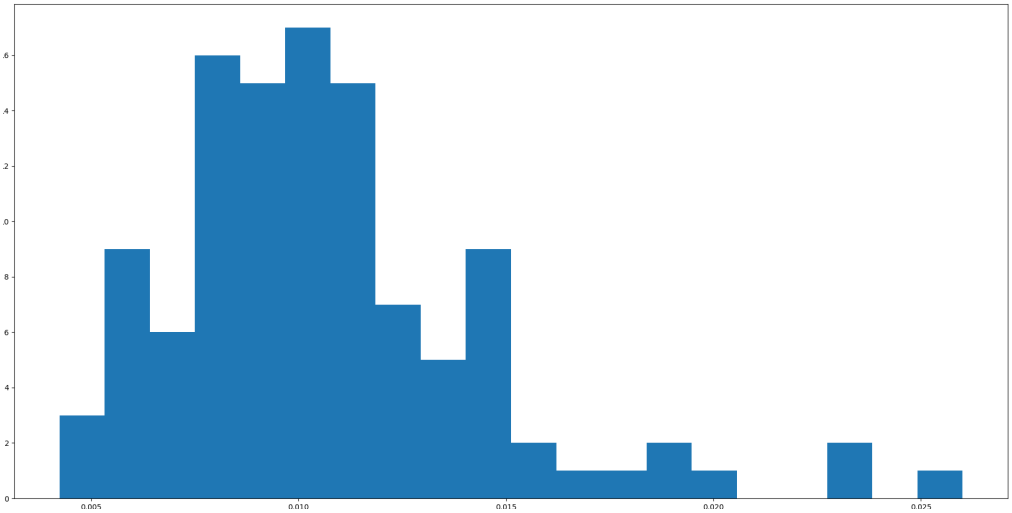


FIGURE 2.5: A histogram of the distortion measured for each of 112 adversarial examples generated using L-BFGS against the VGG16 network on ImageNet images with mean distortion 0.0107

Fast Gradient Sign Method (FGSM) We also implemented a single step attack process uses the gradient of the loss function L with respect to the image to find the adversarial perturbation Goodfellow, Shlens, and Szegedy, 2014. 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 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:

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

This method is faster than L-BFGS and more reliable than FGSM but still produces examples with greater distortion than L-BFGS.

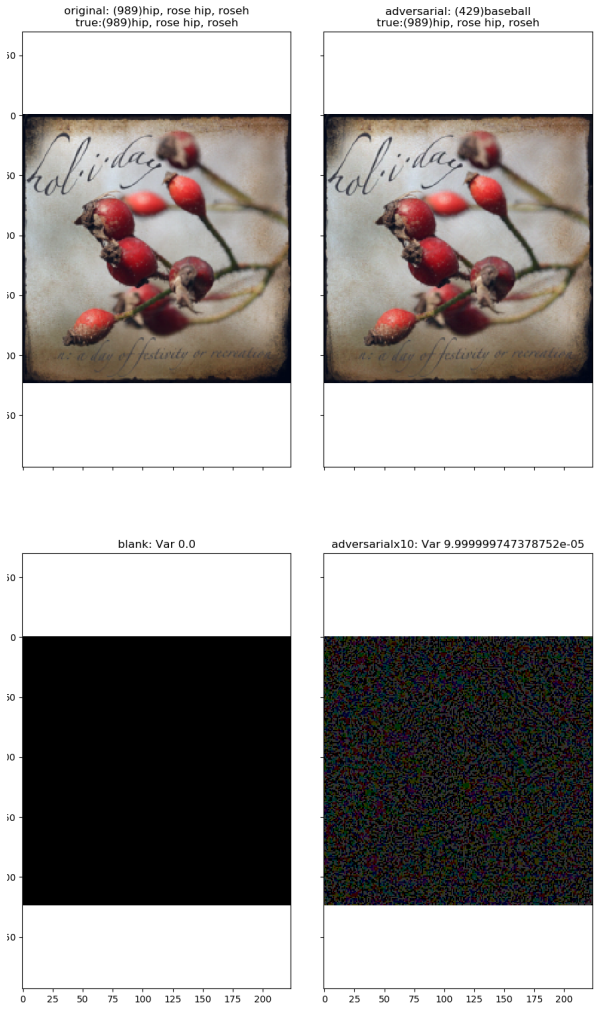


FIGURE 2.6: adversarial example generated against VGG16 (ImageNet) with IGSM. Original Image on the left, adversarial image and added noise (ratio of variance adversarial noise/original image: 0.0000999) on the right.

2.3.2 Other Attacks

The following attack techniques are also prevalent in the literature but have not been replicated in these experiments.

Jacobian-based Saliency Map Attack (JSMA) Another attack noted by **papernot_limitations_2** 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 **wiyatno2018saliency**. 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_deepfool:_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 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 (*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:

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

2.4 Theory of Adversarial Examples

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.

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 **??**. Our statistic is based on the fraction of a Gaussian sampling of the neighborhood of a point which 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.

Chapter 3 Persistent Classification

UNIVERSITY OF ARIZONA

Abstract

David Glickenstein

Department of Mathematics math.arizona.edu

Doctor of Philosophy of Applied Mathematics

A Geometric Framework for Adversarial Vulnerability in Machine Learning

by Brian Bell

Some hypotheses underlying the existence of adversarial examples for classification problems are the high-dimensionality of the data, high codimension in the ambient space of the data manifolds of interest, and/or 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., 2013. 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 wich extends

prior work on smoothed classifiers 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 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.) and we propose three approaches in order to connect robustness with geometric properties of the decision boundary learned by ANNs:

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.

4. We show that robustness tends to correspond with MAG and that directly optimizing a MAG metric improves robustness against linear attacks, however this latter implication does not hold for non-linear adversaries.

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., 2023; 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. ("Adversarial Spheres") 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, buth 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 dynamics and 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. ("A New Defense Against Adversarial Images: Turning a Weakness into a Strength") 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., 2020) 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. ("A New Defense Against Adversarial Images: Turning a Weakness into a Strength") 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 & Bengio study the generalization performance of CNNs by transforming natural image statistics Jo and Bengio, 2017. 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 Kaur, Cohen, and Lipton, 2019. This discovery raises the question of whether this relationship is bidirectional.

Khoury & Hadfield-Menell study the geometry of natural images, and create a lower bound for the number of data points required to cover the manifold Khoury and Hadfield-Menell (2018). Unfortunately, they demonstrate that this lower bound is so large as to be intractable. Shamir et al propose using the tangent space of a generative model as an estimation of this manifold Shamir, Melamed, and BenShmuel, 2021. Magai et al. thoroughly review certain topological properties to demonstrate that neural networks intrinsically use relatively few dimensions of variation during training and evaluation Magai and Ayzenberg, 2022. Vardi et al. demonstrate that even models which satisfy strong conditions related to max margin classifiers are implicitly non-robust Vardi, Yehudai, and Shamir, 2022. 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. ("A New Defense Against Adversarial Images: Turning a Weakness into a Strength"), 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 $\{1, \ldots, \ell\}$). 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 in 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:

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

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 continuous. Indeed, restricted to the pre-image of any set in the power-set, argmax is continuous. Further, we can directly prove that 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

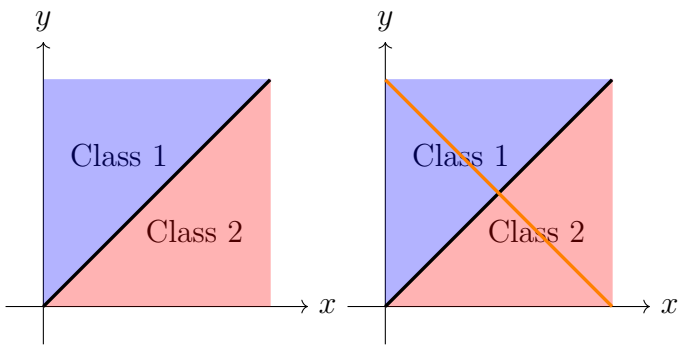


FIGURE 3.1: Decision boundary in $[0,1] \times [0,1]$ (left) and decision boundary restricted to probabilities (right) If the output of F are *probabilities* which add to one, then all points of x will map to the orange line on the right side of Figure 3.1. We note that the point $(0.5, 0.5)$ is therefore 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.

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 $\mathcal{P}(C)$ 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'_f for a classifier $f : \mathbb{R}^N -> \mathcal{C}$ if there exists $\delta > 0$ such that $\forall \epsilon < \delta$, $|f(B_\epsilon(x))| = 1$.

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

Level Set Definition The decision boundary D of a probability valued function f is the pre-image of a union of all level sets of activations $A_c = c_1, c_2, ..., c_k$ defined by a constant c such that for some set of indices L, we have $c = c_i$ for every i in L and $c > c_j$ for every j not in L. 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.

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

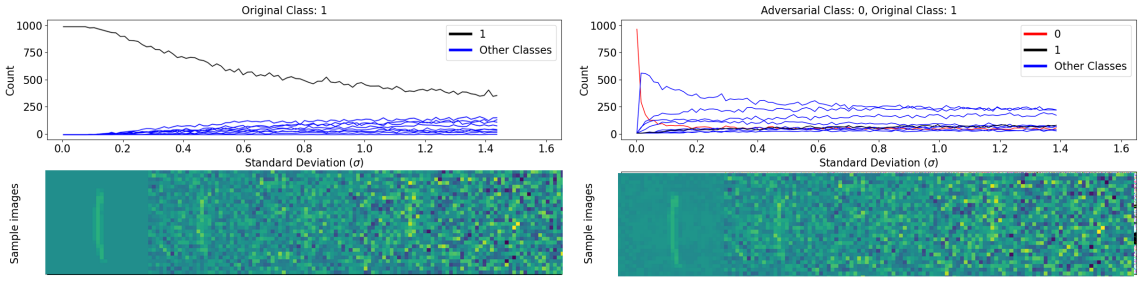


FIGURE 3.2: Frequency of each class in Gaussian samples with increasing variance around a natural image of class `1` (left) and around an adversarial attack of that image targeted at `0` generated using IGSM (right). The adversarial class (`0`) is shown as a red curve. The natural image class (`1`) is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

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.

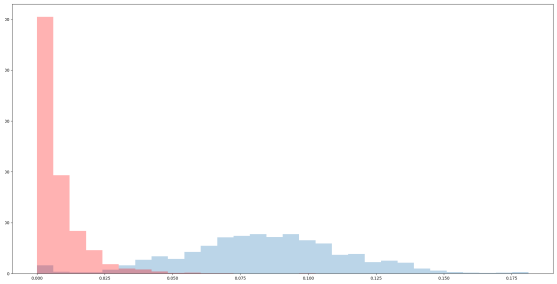


FIGURE 3.3: Histogram of 0.7-persistence of IGSM-based adversarial examples (red) and natural examples (blue) on MNIST.

Next, we investigate the relationship of network complexity and (γ, σ)-stability by revisiting the now classic work of Szegedy et al. (2013) on adversarial examples. Table 3.1 recreates and adds on to part of Szegedy et al., 2013, 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

TABLE 3.1: Recreation of Szegedy et al. (2013, 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

observation is consistent with the idea that networks with more parameters are more 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 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.

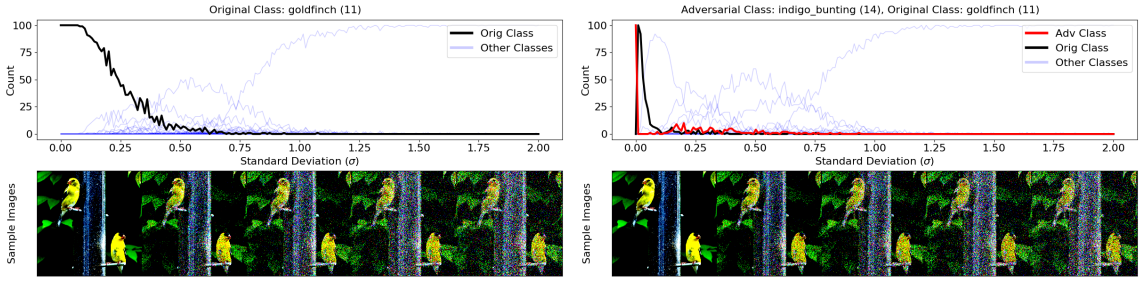


FIGURE 3.4: Frequency of each class in Gaussian samples with increasing variance around a `goldfinch` image (left) and an adversarial example of that image targeted at the `indigo_bunting` class and calculated using the BIM attack (right). Bottoms show example sample images at different standard deviations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

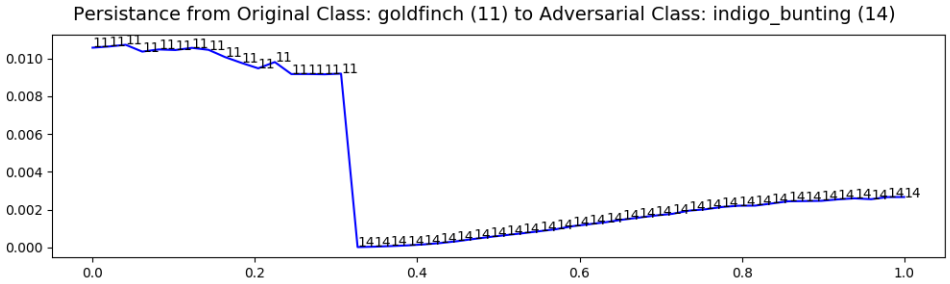


FIGURE 3.5: The 0.7-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) on a vgg16 classifier. The classification of each image on the straight line is listed as a number so that it is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

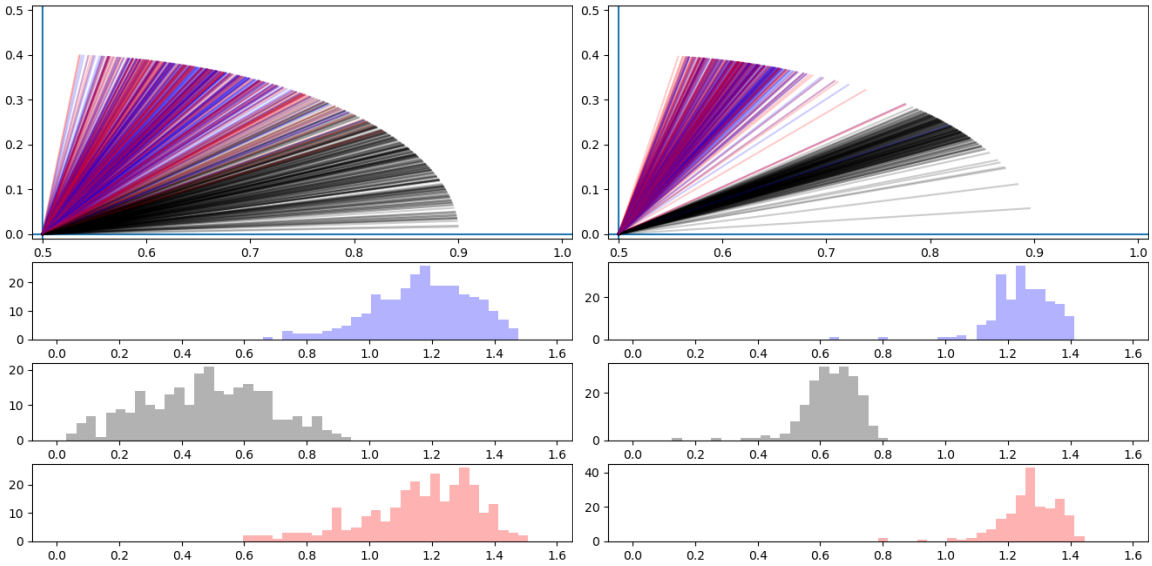


FIGURE 3.6: Decision boundary incident angles between test and test images (left) and between test and adversarial images (right). Angles (plotted Top) are referenced to decision boundary so $\pi/2$ radians (right limit of plots) corresponds with perfect orthogoonality to decision boundary. Lines and histograms measure angles of training gradients (Blue) linear interpolant (Black) and adversarial gradients (Red)

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 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 less acute. This suggests that adversaries are exploiting the obliqueness of the decision boundary with respect to test points. 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, 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 $\{z \times \mathbf{W}^T : z \in \mathbb{R}^k\}$ 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.

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

3.4.5 Manifold Aligned Gradients

Component vectors extracted from the original dataset are used to project gradient examples onto our pre-defined image manifold. 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 we will refer to as $P_\mathcal{M}$. The norm 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 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.

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

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) and projected gradient descent (PGD) (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. “Robust" in our case refers to adversarial training. All robust models were trained using the l_∞ norm at $\epsilon = 0.1$. Manifold Robust refers to both optimizing our manifold objective and robust training simultaneously.

Figure 3.7 shows the cosine similarity on the testing set of PMNIST for both the Manifold model and Robust model. Higher values indicate the model is more aligned with the manifold. Both 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 robustness 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 exploit oblique structure of

overlapping decision boundaries around the adversarial class, whereas natural images lie outside such regions.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 obliqueness of the boundary.

We reinforced this obliqueness hypothesis by computing angles of linear interpolants with respect to the decision boundary, showing that most interpolants cross the decision boundary between classes at very shallow angles. Furthermore, adversarial interpolants tend to cross at less shallow, but still acute angles. Furthermore, we present the simplest possible case of our hypothesis that manifold alignment implies adversarial robustness. Extending this to show results on more complex models and datasets is left to future work. In this early work, we only test against a linear manifold and show that it provides robustness against FGSM. We conclude that training a model to be aligned with a low dimensional manifold on which your data lies is related to robust training. While this model shows some properties of adversarial robustness, it is still vulnerable to PGD attacks. Additionally, a model trained to be robust using adversarial training shows manifold alignment under our definition.

3.6 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., 2018; "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 "On Adaptive Attacks to Adversarial Example Defenses".

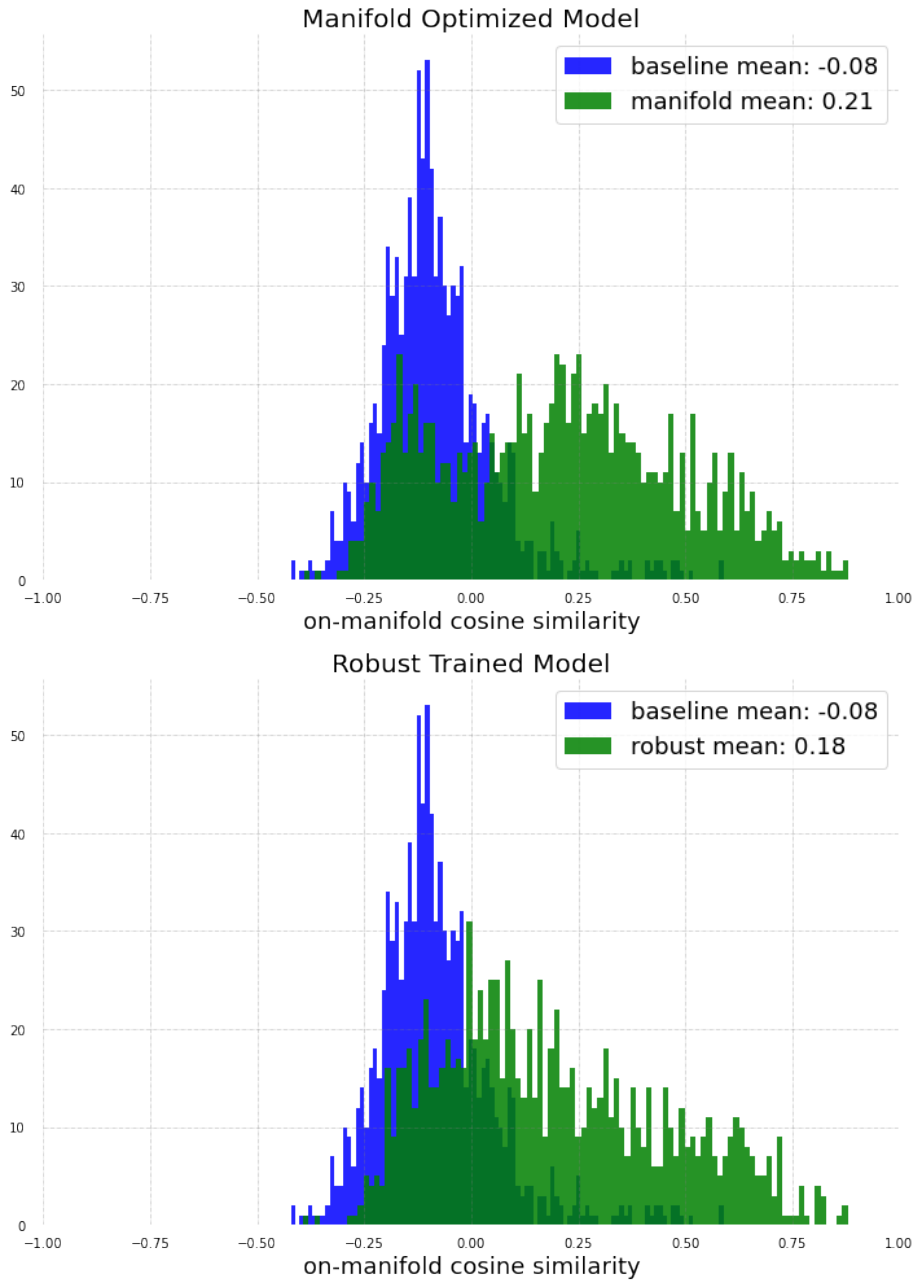


FIGURE 3.7: Comparison of on-manifold components between baseline network, robust trained models, and manifold optimized models. Large values indicate higher similarity to the manifold. Both robust and manifold optimized models are more 'on-manifold' than the baseline, with adversarial training being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

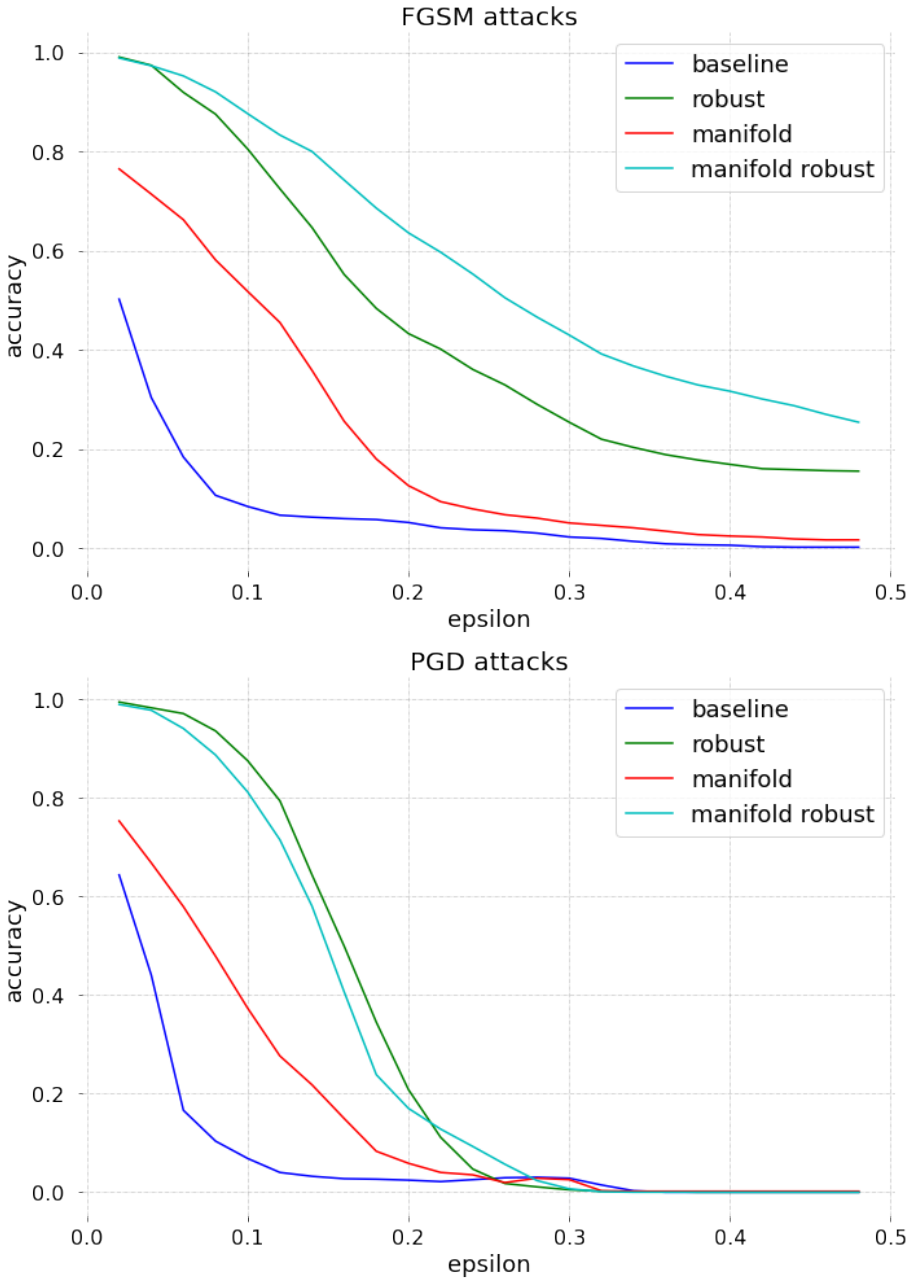


FIGURE 3.8: Comparison of adversarial robustness for PMNIST models under various training conditions. For both FGSM and PGD, we see a slight increase in robustness from using manifold optimization. Adversarial training still improves performance significantly more than manifold optimization. Another 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 $= 0.1$.

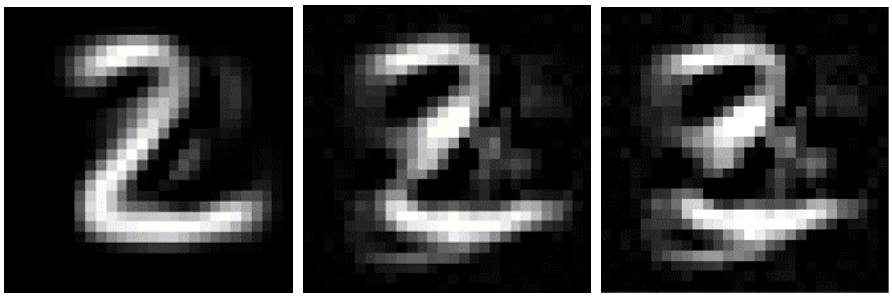


FIGURE 3.9: Visual example of manifold optimized model transforming 2 into 3. Original PMNIST image on left, center image is center point between original and attacked, on right is the attacked image. Transformation performed using PGD using the l_∞ norm. Visual evidence of manifold alignment is often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

Chapter 4 Kernel Neural Equialence

UNIVERSITY OF ARIZONA

Abstract

David Glickenstein

Department of Mathematics math.arizona.edu

Doctor of Philosophy of Applied Mathematics

A Geometric Framework for Adversarial Vulnerability in Machine Learning

by Brian Bell

We explore the equivalence between neural networks and kernel methods by deriving the first exact representation of any finite-size parametric classification model trained with gradient descent as a kernel machine. We compare our exact representation to the well-known Neural Tangent Kernel (NTK) and discuss approximation error relative to the NTK and other non-exact path kernel formulations. We experimentally demonstrate that the kernel can be computed for realistic networks up to machine precision. We use this exact kernel to show that our theoretical contribution can provide useful insights into the predictions made by neural networks, particularly the way in which they generalize.

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.

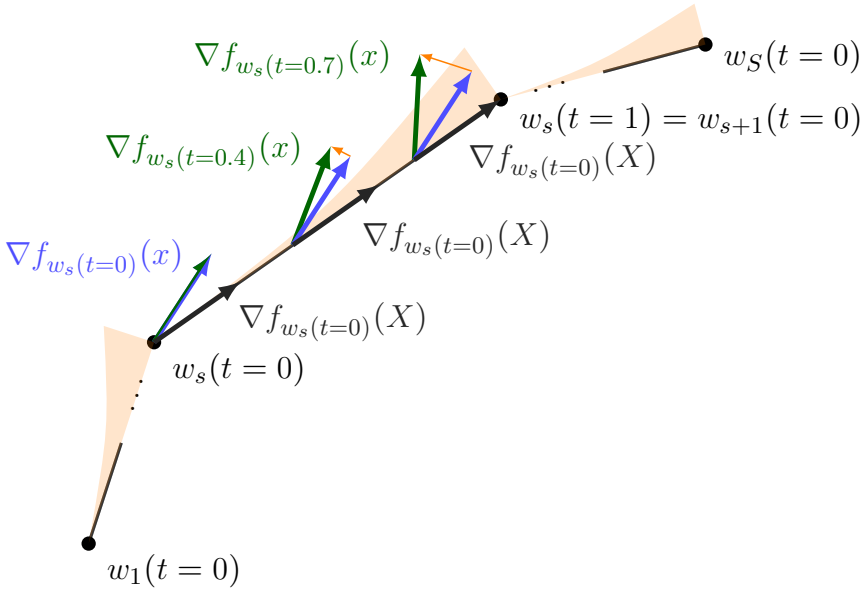


FIGURE 4.1: Comparison of test gradients used by Discrete Path Kernel (DPK) from prior work (Blue) and the Exact Path Kernel (EPK) proposed in this work (green) versus total training vectors (black) used for both kernel formulations along a discrete training path with S steps. Orange shading indicates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

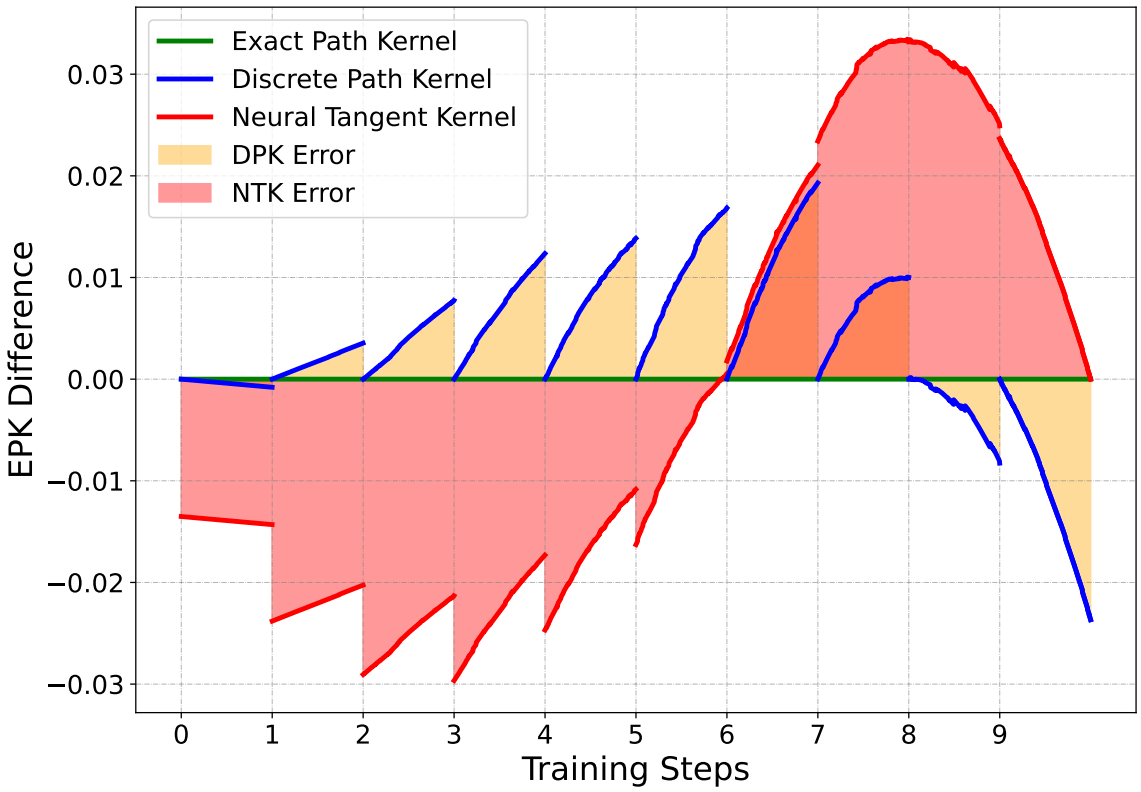


FIGURE 4.2: Measurement of gradient alignment on test points across the training path. The EPK is used as a frame of reference. The y-axis is exactly the difference between the EPK and other representations. For example $EPK - DPK = \langle \phi_{s,t}(X), \phi_{s,t}(x) - \phi_{s,0}(x) \rangle$ (See Definition 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.

There has recently been a surge of interest in the connection between neural networks and kernel methods **bietti2019bias**; **du2019graphntk**; **tancik2020fourierfeatures**; **abdar2021uq**; **geifman2020similarity**; **chen2020generalized**; **alemohammad2021recurrent** Much of this work has been motivated by the the neural tangent kernel (NTK), which describes the training dynamics of neural networks in the infinite limit of network width **jacot2018neural**. We argue that many intriguing behaviors arise in the *finite*

parameter regime **DBLP:conf/nips/BubeckS21**. 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 **ghojogh2021**; **shawe2004kernel**; **zhao2005extracting**.

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 **incudini2022quantum** due to dependence of the kernel on test data unless strong conditions are imposed on the training process as by **chen2021equivalence**. 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 **DBLP:conf/iclr/LeeBNSPS18**; **chen2021equivalence**. While the properties of infinite neural networks are fairly well understood **neal1996priors**, 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 **domingos2020every**; **chen2021equivalence**.

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 **jacot2018neural**. In this setting, it has been shown that neural networks are equivalent to support vector machines, drawing a connection to maximum margin classifiers **chen2021equivalence**; **chizat2020maxmargin**. Similarly, Shah et al. 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 Shah, Jain, and Netrapalli, 2021.

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 $\varepsilon \to 0$) **domingos2020**. 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 **incudini2022quantum**. Moreover, the formulation of the sample weights and bias term in the DPK depends on its test points. Chen et al. 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 **chen2021equivalence**.

We address the limitations of **domingos2020** and **chen2021equivalence** 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. **chen2021equivalence** 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*

$$K(x) = b + \sum_{i \in I} a_i k(x, x_i) \tag{4.1}$$

where the $a_i \in \mathbb{R}$ do not depend on x, $b \in \mathbb{R}$ is a constant, and k is a kernel. **rasmussen2006gaussian**

By Mercer's theorem **ghojogh2021** *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.*

$$K(x) = b + \sum_{i \in I} a_i \langle \phi(x), \phi(x_i) \rangle \tag{4.2}$$

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 $\epsilon \not\to 0$ 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 $x \in H \times \{0\}$ 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*

$$K_{EPK}(x, x', s) = \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x') \rangle dt \tag{4.3}$$

where

$$\phi_{s,t}(x) = \nabla_w f_{w_s(t,x)}(x) \tag{4.4}$$

$$w_s(t) = w_s - t(w_s - w_{s+1}) \tag{4.5}$$

$$w_s(t,x) = \begin{cases} w_s(0), & \text{if } x_I = 1 \\ w_s(t), & \text{if } x_I = 0 \end{cases} \tag{4.6}$$

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 C.1.1). 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.*

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:*

$$f_{w_N} = KE(x) := \sum_{s=1}^{N} \sum_{i=1}^{M} a_{i,s} K_{EPK}(x, x', s) + b \tag{4.7}$$

where

$$a_{i,s} = -\varepsilon \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \tag{4.8}$$

$$b = f_{w_0}(x) \tag{4.9}$$

Proof Sketch. Assuming the theorem hypothesis, we'll measure the change in model output as we interpolate across each training step s by measuring the change in model state along a linear parametrization $w_s(t) = w_s - t(w_s - w_{s+1})$. We will let d denote the number of parameters of f_w. For brevity, we define $L(x_i, y_i) = l(f_{w_s(0)}(x_i), y_i)$ where l is the loss function used to train the model.

$$\frac{d\hat{y}}{dt} = \sum_{j=1}^{d} \frac{d\hat{y}}{\partial w_j} \frac{dw_j}{dt} \tag{4.10}$$

$$= \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{M} \frac{\partial L(x_i, y_i)}{\partial f_{w_s(0)}(x_i)} \frac{\partial f_{w_s(0)}(x_i)}{\partial w_j} \right) \tag{4.11}$$

We use fundamental theorem of calculus to integrate this equation from step s to step $s+1$ and then add up across all steps. See Appendix C.1.1 for the full proof. □

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:*

$$f_{w_N}(x) = KM(x) := b + \sum_{i=1}^{M} a_{i,0} K_{NEPK}(x, x') \tag{4.12}$$

See Appendix C.1.2 for full proof. By combining theorems 4.3.6 and 4.3.7, 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

$\phi_{s,t}(x)$ depends on both s and t, which is non-standard but valid, however an important consequence of this mapping 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:

$$k'(x, x') = \langle \nabla_w f_{w_s(t)}(x), \nabla_w f_{w_s(0)}(x') \rangle \tag{4.13}$$

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

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt = \nabla_w f_{w_s(0)}(x) \tag{4.14}$$

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

$$\frac{\partial w_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(x_i), y_i) \tag{4.15}$$

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

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt \approx \nabla_w f_{w_s(0)}(x) \tag{4.16}$$

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 4.15 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.

4.3.4 Ensemble Reduction

In order to reduce the ensemble representation of Equation (4.7) to the kernel representation of Equation (4.12), we require that the sum over steps still retain the properties of the kernel (symmetry and positive semi-definiteness). In particular we

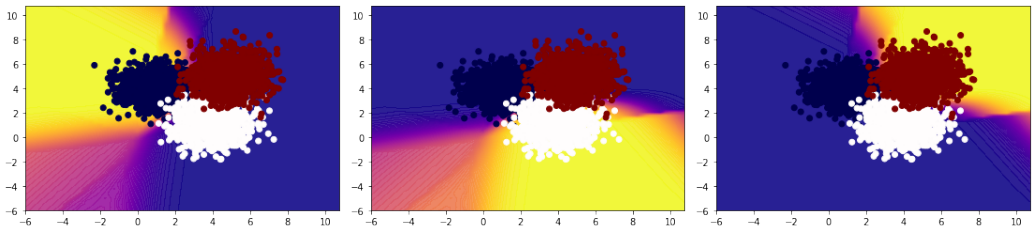


FIGURE 4.3: Updated predictions with kernel a_i updated via gradient descent with training data overlaid for classes 1 (left), 2 (middle), and 3 (right). The high prediction confidence in regions far from training points demonstrates that the learned kernel is non-stationary.

require that for every subset of the training data x_i and arbitrary α_i and α_j, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{M}\sum_{s=1}^{N}\alpha_i\alpha_j a_{l,s}\int_0^1 K_{\text{EPK}}(x_i,x_j)dt \geq 0 \tag{4.17}$$

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.1.2). 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. **chen2021equivalence**, however the kernel that they derive for a finite-width case is of the form

$$K(x, x_i) = \int_0^T |\nabla_f L(f_t(x_i), y_i)| \langle \nabla_w f_t(x), \nabla_w f_t(x_i) \rangle dt \tag{4.18}$$

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.

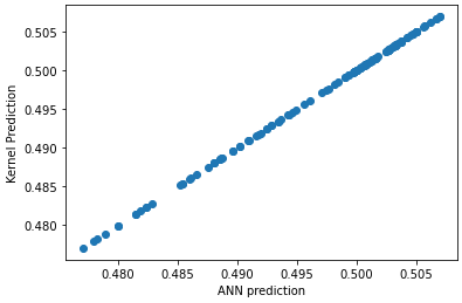


FIGURE 4.4: Class 1 EPK Kernel Prediction (Y) versus neural network prediction (X) for 100 test points, demonstrating extremely close agreement.

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 4.3 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 **rasmussen2006gaussian** 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 (**chizat2020maxmargin** 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 **pytorch2019**. 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 (**chen2021equivalence**, **domingos2020**, **incudini2022quantum**) 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 **simonyan2013deep**; **lundberg2017unified**; **Selvaraju__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 **chizat2020maxmargin**. As stated in previous work **domingos2020**, 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, $a_{i,s} = -\varepsilon \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial f_i}$, 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 **hardle2004nonparametric**.

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

NIPS2005_663772ea. We're hopeful about the scalability of learned kernels based on recent results in scaling kernel methods **snelson2005sparse**. 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 **szegedy2013intriguing**; 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.

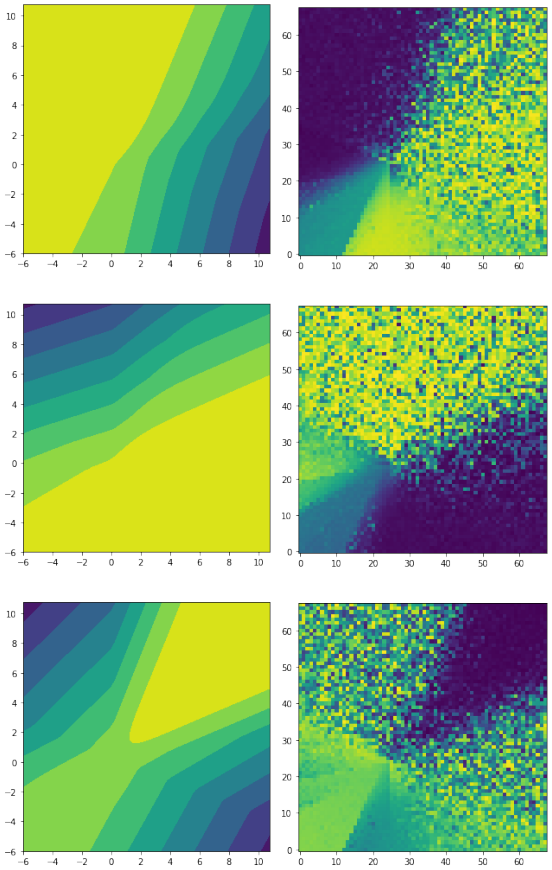


FIGURE 4.5: (left) Kernel values measured on a grid around the training set for our 2D problem. Bright yellow means high kernel value (right) Monte-Carlo estimated standard deviation based on gram matrices generated using our kernel for the same grid as the kernel values. Yellow means high standard deviation,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

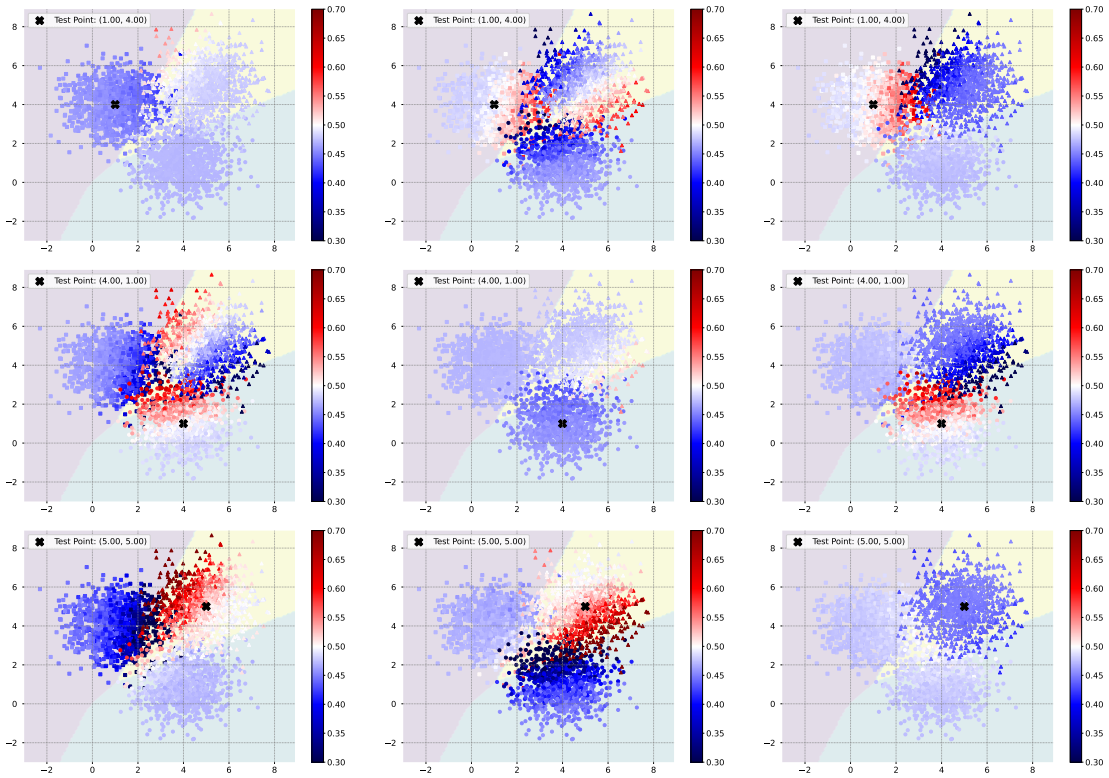
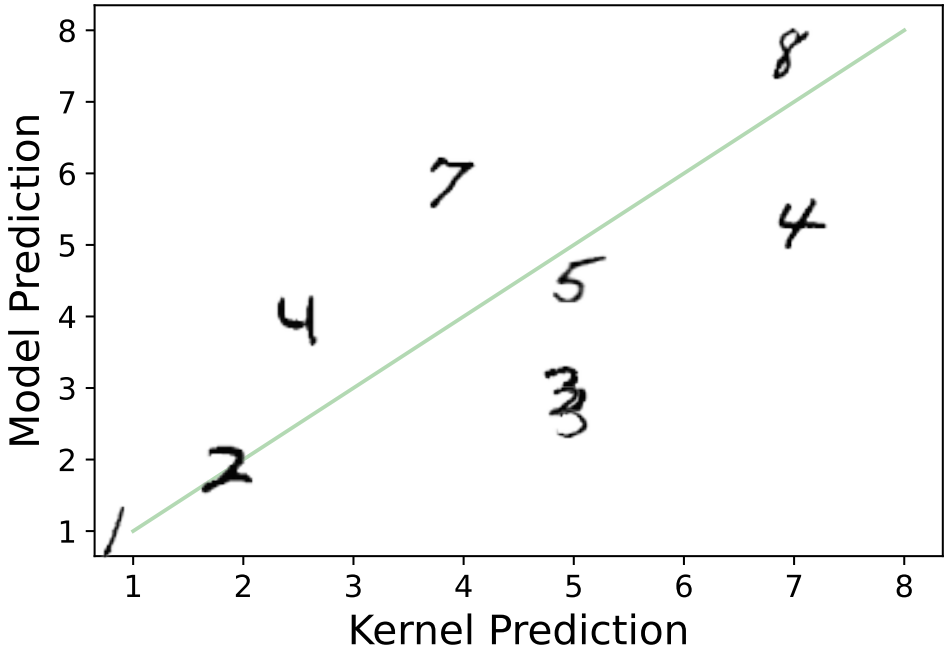
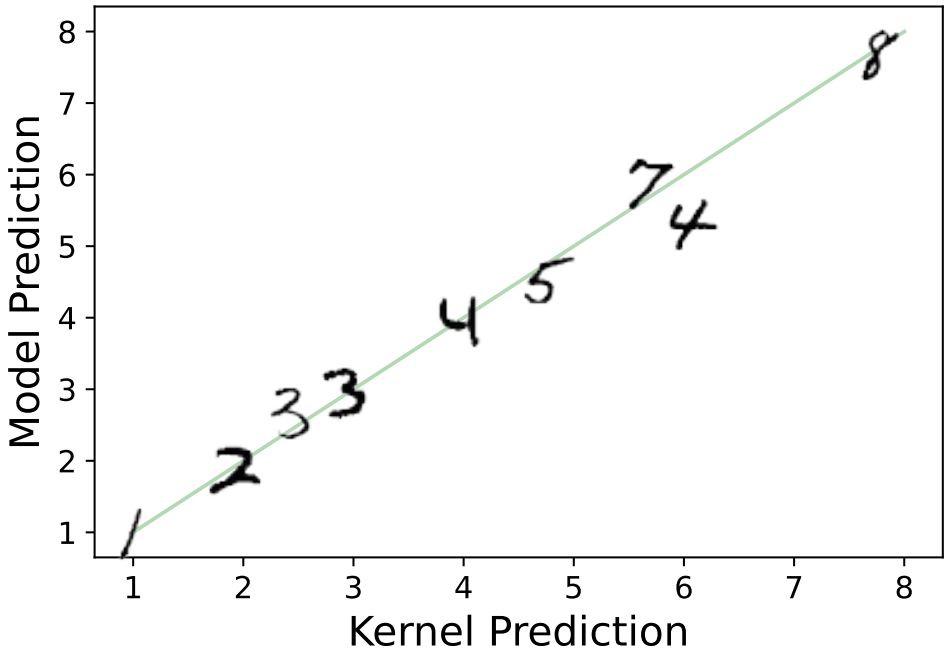
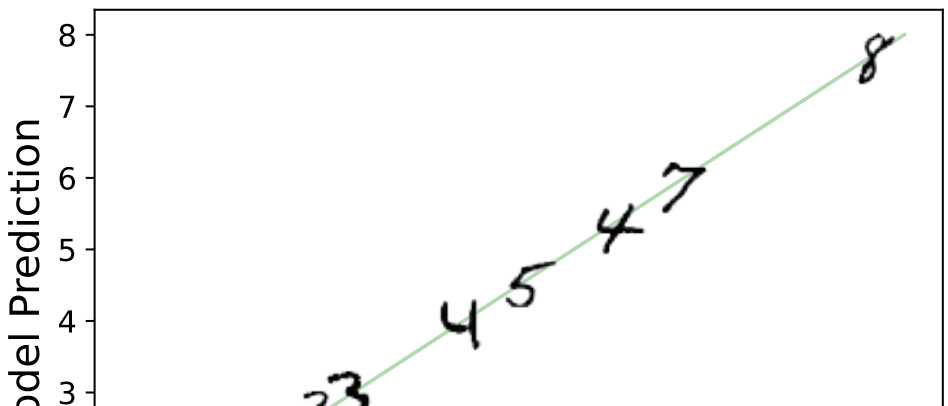


FIGURE 4.6: Plots showing kernel values for each training point relative to a test point. Because our kernel is replicating the output of a network, there are three kernel values per sample on a three class problem. This plot shows kernel values for all three classes across three different test points 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

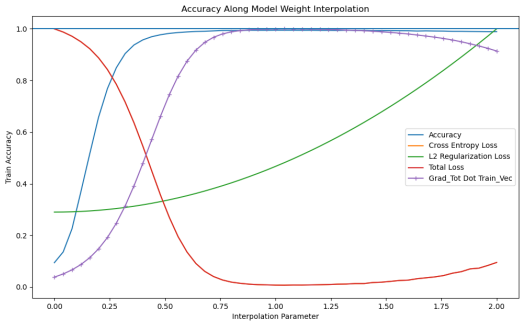


FIGURE 4.8: This plot shows a linear interpolation $w(t) = w_0 + t(w_1 - w_0)$ of model parameters w for a convolutional neural network f_w from their starting random state w_0 to their ending trained state w_1. The hatched purple line shows the dot product of the sum of the gradient over the training 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 Decision Boundaries

5.1 decision boundary definitions

5.2 The Argmax Function

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:

$$\text{argmax} : \mathbb{R}^k \to \mathcal{P}(C) \tag{5.1}$$

Defined this way, we cannot necessarily consider arg max 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 continuous.

Indeed, restricted to the pre-image of any set in the power-set, argmax is continuous.

Further, we can directly prove that 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 $\mathcal{P}(C)$ is a closed set. We will call this closed set the argmax Decision Boundary.

Note : there are ways argmax can be forced to break ties, i.e. by ordering the

Questions:

Is the decision boundary connected

5.3 Defining Decision Boundaries

5.3.1 Complement Definition

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

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

5.3.2 Constructive Definition

A point x is on the *decision boundary* D of a classifier f if $\forall \epsilon > 0$, $|f(B_\epsilon(x))| \geq 2$.

A point x is a *binary decision point* if $\exists\delta > 0$ such that $\forall\epsilon > 0$ if $\epsilon < \delta$, then $|f(B_\epsilon(x))| = 2$.

A point x is on the *K-decision boundary* D^K of a classifier f if $\exists\delta > 0$ such that $\forall\epsilon > 0$ if $\epsilon < \delta$, then $|f(B_\epsilon(x))| = K$.

5.3.3 Level Set Definition

The decision boundary D of a probability valued function F is a union of all level sets $L_{c_1,c_2,a}$ defined by two classes c_1 and c_2 and a constant a which satisfy two properties: First, given $x \in L_{c_1,c_2,a}$, $F(x)_{c_1} = F(x)_{c_2} = a$ where a is a constant also defining the level set. Second, for all $c \notin \{c_1, c_2\}$, we have $a > F(x)_c$.

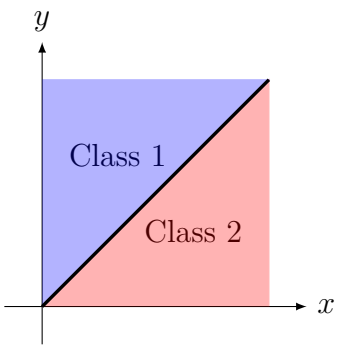
5.3.4 Images of Decision Boundaries

We can consider a classification workflow f from a data space X (e.g. $\mathbb{R}^N$) using a model F mapping the data space X to a probability space Y (e.g. $[0,1]^K$ where K is the number of classes) and using argmax to convert continuous representations in Y to discrete classes in the set of classes $\mathbb{C}$.

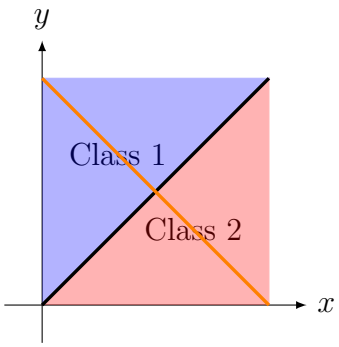
$$X \xrightarrow{F} Y \xrightarrow{\text{argmax}} \mathcal{C} \tag{5.2}$$

In this way we define a usual discrete classifier $f = F \circ \text{argmax}$

The decision boundary D is defined on X, however we may look at the image of the decision boundary under F. For example, if there are only two classes, then $Y = [0,1]^2$ and the decision boundary can be nicely visualized by the black line below:



Furthermore, if the output of F are *probabilities* which add to one, then all points of x will map to the orange line:



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.

5.4 Properties of Decision Boundaries

In practice, decision boundaries have a variety of properties. The decision boundary is a pre-image under an onto function that is not 1-1, which allows decision boundaries

to vary in upto as many dimensions as the data space. In practice, this will likely be many fewer. In particular, decision boundaries should correspond with the structure of the data. If the data is embedded within a k-manifold within the input space, the decision boundary can be though of on the dual space of that manifold.

The dual space of a k-manifold embedded in an n-dimensional Euclidean space is the space of differential k-forms on the manifold. More precisely, for a k-manifold M embedded in an n-dimensional Euclidean space, the dual space is defined as the space of covariant k-tensors on M, denoted by $\Lambda^k(M)$, which consists of all linear maps from the tangent space of M at each point to the real numbers.

In particular, when M is a smooth k-dimensional manifold, $\Lambda^k(M)$ is the space of smooth k-forms on M, which can be locally expressed as linear combinations of differential forms of the form $dx_I = dx_{i_1} \wedge dx_{i_2} \wedge \cdots \wedge dx_{i_k}$, where the indices $i_1 < i_2 < \cdots < i_k$ denote the coordinates of a k-dimensional chart on M and dx_i denotes the differential of the i-th coordinate function.

Intuitively, a k-form on M assigns to each k-dimensional oriented subspace of the tangent space at a point a signed volume, and the space of all such k-forms forms the dual space of M. This dual space is important in various areas of mathematics and physics, including differential geometry, topology, and gauge theory.

5.4.1 Delaunay Decision Boundaries

The simplest special case within this dual space is a classifier which is defined using the Voronoi diagram for the dataset. We can use the voronoi diagram to define a classifier for an arbitrary query point by simply assigning it the class of the training

point whose voronoi cell contains it. This has several useful properties, though one in particular stands out.

Given two points x and y in the training set X which define a delaunay triangulation and its corresponding voronoi diagram (which is the dual of the delaunay triangulation) we can see that a line connecting x to y must orthogonally cross the decision boundary – which is exactly the voronoi diagram of this dataset.

Proof:

Proof. Let x and y be two points in a dataset X that share an edge in the Delaunay triangulation for X, and let e be the edge they share in the Delaunay graph. Suppose that e is not orthogonal to the face of the Voronoi diagram that x and y share. Then there exists a point z in the interior of the Voronoi cell for x and y such that the line through z and the midpoint of e intersects the face of the Voronoi diagram at a point p.

Since z is in the Voronoi cell for x and y, it follows that $d(z, x) < d(z, y)$ (where d is the Euclidean distance). But this contradicts the fact that x and y share an edge in the Delaunay triangulation, which implies that the circumcircle of the triangle formed by x, y, and z does not contain any other points in X. In particular, this means that the circumcenter of this triangle lies on the perpendicular bisector of e, which implies that e is orthogonal to the face of the Voronoi diagram that x and y share.

Therefore, we have shown that if x and y share an edge in the Delaunay triangulation for X, then the edge they share in the Delaunay graph must be orthogonal to the face of the Voronoi diagram that they share. □

It is immediately clear that for points not in the training set, the line connecting them with eachother or with a point in the training set need not cross orthogonal

to the voronoi cell boundaries. In order to talk clearly about such non-orthogonal crossings, we will define

Definition 5.4.1. *In arbitrary dimensional space, the* angle of incidence *between a line and a plane is defined as the angle between the line and its projection onto the plane, measured in the plane.*

More precisely, let $\mathbb{R}^n$ be an n-dimensional Euclidean space, let ℓ be a line in $\mathbb{R}^n$ with direction vector $\vec{v}$, and let P be a plane in $\mathbb{R}^n$ with normal vector $\vec{n}$. Suppose that ℓ intersects P at a point Q. Let H be the orthogonal projection of ℓ onto P, and let θ be the angle between ℓ and H.

Then, the angle of incidence between ℓ and P is defined as $\alpha = \frac{\pi}{2} - \theta$, where π is the constant representing the ratio of the circumference of a circle to its diameter.

Note that when $n = 3$, this definition reduces to the familiar definition of the angle of incidence between a line and a plane in three-dimensional space. However, the definition is valid for arbitrary dimensional space.

and

Definition 5.4.2. *The*

5.4.2 Level Set Definition

Definition 5.4.3. *The* decision boundary *D of a probability valued function F is a union of all level sets $L_{c_1,c_2,a}$ defined by two classes c_1 and c_2 and a constant a which satisfy two properties: First, given $x \in L_{c_1,c_2,a}$, $F(x)_{c_1} = F(x)_{c_2} = a$ where a is a constant also defining the level set. Second, for all $c \notin \{c_1, c_2\}$, we have $a > F(x)_c$. This is the* decision level set *for level a. For a given point x on the decision boundary*

of a function, the decision level *is the value a defining the level set of the decision boundary of which x is a member.*

Note: *Any point x can be in at most one level set of the decision boundary.*

Definition 5.4.4. *The* dimension *of the decision boundary at a point x is the number of dimensions needed to span $\{z : (x + z)$ is in the decision level a of the level set containing $x\}$*

A primary property of interest in adversariall attacks is "sharpness" i.e. the property that a partition of a set which is attributed to one class might stab needle-like into a partition of another set. We will investigate this "sharpness" concept by examining both the incident angle (the acuteness of angles in the decision boundary) and the dimension of the decision boundary, since both properties behave like one another.

There is one conditional bound which is specifically for voronoi cells which are not exterior to the voronoi diagram. Such cells are bounded. The line connecting a point within a bounded cell to the point defining any adjacent voronoi cell.

5.4.3 Weighted Delaunay Decision Boundaries

This simple voronoi classifier has the limitation that it only cares about which individual training datapoint is closest to the query point, but is not sensitive to the distribution of this data. If we suppose that this data is distributed on a manifold with significant co-dimension with respect to the ambient data space, then it would be best to use a metric based on this manifold, so that distance among the data, and hence the delaunay triangulation and the comensurate voronoi cells would be

determined by this metric and thus properly sensitive to the underlying geometry of this distribution. Alas, this is akin to having the generating function for nature. It cannot be computed. So instead, we can take advantage of the Delaunay triangulation with the usual euclidean metric as an approximation of the lower dimensional manifold in this case.

This method is limited by the quality of the training dataset as a sampling of the distribution in question. The Delaunay triangulation is very sensitive to outliers from a distribution, meaning that our samples must have relatively little noise in order to be treated as appropriate samples from the low-dimensional manifold. Second, Delaunay triangulations are unstable in high-dimensions where very small triangles are more likely. Both of these problems are lessened by using data which are sufficiently plentiful relative to the dimension of the lower dimensional manifold and by requiring that they do not have more than a small amount of noise to perturb them from the low dimensional manifold.

Suppose the lower dimensional manifold M could be defined, and the training dataset X could be projected onto M. The decision boundary partitioning M would be a set of slices orthogonal to M which would partition any training dataset X into its appropriate parts. In particular, for a sufficiently dense sampling x, we could define this decision boundary by examining the data points immediately on either side the decision boundary wherever they meet. We can define an approximation of the decision boundary as the orthogonal partition that we expect to be equidistant between adjacent points of different classes for all samplings of training points. In fact, we can define this boundary in the limit using delaunay triangulation:

Definition 5.4.5. *Let M be a low dimensional manifold and let X_ε be a sampling of*

*points on M for which the maximal edge length along its Delaunay triangulation D_ε is ε. Given a query point z, the $\varepsilon-$Delaunay weight for class i, $w(z)_i$, is the sum of the barycentric coordinates of z relative to the vertices with label i of the simplex of D_ε which contains z. The weight vector ε boundary level set for factor a, $B_{\varepsilon,a}$ is the level set $\{z \in M : \text{for all } i, \text{ we have } w(z)_i \leq a \text{ and for some } i, j, \ w(z)_i = w(z)_j = a\}$. The union of all such level sets over a is the $\varepsilon-$*Delaunay-Boundary B_ε.

Remark, we can extend the decision boundary outside of M by simply extending the Delaunay-Boundary using the usual euclidean metric.

Theorem 5.4.6. *Given a low dimensional manifold M,*

$$\lim_{\varepsilon \to 0} B_\varepsilon(M) = B(M) \tag{5.3}$$

$B(M)$ is the decision boundary restricted to M.

In this way, a particular training set X defines an approximation of the general Delaunay Decision Boundary which is well defined outside of M. If the decision boundary is smooth, then for a sufficiently dense training set X which is sufficiently close to M, this approximation can be bounded by ε and a lipshitz constant coming from the mean value theorem on the decision boundary.

It is this approximate form of the decision boundary which we may in practice evaluate.

5.4.4 Orthant and Wedge

In order to study decision boundaries, we must talk about properties related to how decision boundaries interact. We desire a test-object whose dimensionality and

acuteness of angle we can control. To this end we will use the $k-$dimensional orthant:

Definition 5.4.7. *Given an integer k, the k-*orthant *in n dimensions is the set $\{x \in \mathbb{R}^n$ such that for every natural number $i \leq k$, $x_i > 0\}$*

This concept for an orthant can be extended to be acute in arbitrarily many angles by tipping toward x to form a wedge.

Definition 5.4.8. *Given integer k and a set of $k-1$ angles Θ, the $k-\Theta-$*wedge *in n dimensions is the set $\{x \in \mathbb{R}^n$ such that $x_1 > 0$ and for every natural number $1 < i \leq k$, $x_i > x_1 \tan(\theta_{i-1})\}$.*

These objects allow us to control the dimensionality and acuteness of a decision boundary to test decision boundary metrics. Another recipe for analysis is standardizing the paths along which we will cross decision boundaries. We will use two general paths relative to the wedge for comparison: centerline and $x-$parallel:

Definition 5.4.9. *The* centerline *$k-\Theta-$wedge crossing is defined along the line $x_1 = t$, for $1 < i \leq k$ $x_i = t\tan(\theta_{i-1}/2)$, and for $i > k$ $x_i = 0$.*

*The $x-\varepsilon-$*parallel *$k-\Theta-$wedge crossing is defined along the line $x_1 = t$, for $i > 1$ $x_i = \varepsilon$*

Although we can easily determine whether a point is inside or outside the wedge simply by checking whether it satisfies the conditions defining the wedge, we also wish to compute the projection or nearest point on the wedge for an arbitrary query point. This involves solving for the closest point to a convex object, which can be accomplished by projection onto convex sets.

Computing this projection requires we determine which inequalities have been violated. Each inequality corresponds with a hyper-plane.

Definition 5.4.10. *Given a finite set of half-planes Y_i in $\mathbb{R}^n$ and a point x, let I contain all i such that $x \notin Y_i$. Then since the intersection of any number of convex sets is convex, $Y_x = \cap_{i \in I} Y_i$ is a convex set. Furthermore, there is exactly one point z with $|x - z| \leq |x - y|$ for every y in Y_x. This is the* projection *of x onto Y_x.*

In order to compute this point in practice, we must first determine all half-planes that do not contain x and then perform orthogonal projection in sequence along these half-planes using Projection onto Convex Sets /citevonneuman-pocs.

Lemma 5.4.11. *If all half-planes Y_i with corresponding normal vector v_i have boundaries $B(Y_i)$ which are mutually orthogonal, i.e. if $\langle v_i, v_j \rangle = 0$ for every combination of i and j, then the nearest point can be solved by iterative projection*

$$proj_{\mathbf{v_{i_1}}}(proj_{\mathbf{v_{i_2}}}(proj_{\mathbf{v_{i_3}}}...(\mathbf{x}))) \tag{5.4}$$

(**vonneuman-pocs**)

In the case of the upper wedge, the boundaries are not mutually orthogonal, so simple orthogonal projection is not sufficient. We must explore this projection in more detail. Really, what we wish to do is project the query point x onto the intersection of all of its violated boundaries. Iteratively, we can orthogonaly project onto the first hyperplane. We will conduct our projection by subtracting a projection onto a normal vector for each hyperplane. Given a point x, a hyperplane p_i and its normal vector v_i,

we define

$$proj_{\mathbf{p_i}}(\mathbf{x}) = x - x \cdot v_i \tag{5.5}$$

The normal vectors for our wedge come in two forms. For the orthant, all of the normals are of the form $v_i = e_i$. For the angled wedge planes, they are all of the form $w_i = e_i \cos(\theta_i) - e_1 \sin(\theta_i)$ where $i > 1$. We immediately note that $v_i \cdot v_j = 0$ for every natural number i and j less than the number of planes k. However,

$$w_i \cdot w_j = \langle e_i \cos(\theta_i) - e_1 \sin(\theta_i), e_j \cos(\theta_j) - e_1 \sin(\theta_j) \rangle \tag{5.6}$$

$$= e_i e_j \cos(\theta_i)\cos(\theta_j) - e_1 e_j \cos(\theta_j)\sin(\theta_j) - e_1 e_i \cos(\theta_i)\sin(\theta_j) + e_1^2 \sin(\theta_j)\sin(\theta_i) \tag{5.7}$$

$$= e_1 \sin(\theta_j)\sin(\theta_i) \tag{5.8}$$

$$\tag{5.9}$$

This means that none of the w_j are orthogonal to each-other and furthermore

$$v_1 \cdot w_i = \langle e_1, -e_1 \sin(\theta_i) + e_i \cos(\theta_i) \rangle \tag{5.10}$$

$$= -\sin(\theta_i) \tag{5.11}$$

We will leave this latter lack or orthogonality for last, and start by determining how to sequentially orthogonally project onto the w_j. Critically, we need only project onto the intersection of each w_i with all previously projected w_j. We begin by projecting from within the plane defined by w_i onto the plane defined by w_j. These vectors are not orthogonal, so we must derive a new plane and its normal vector w_{ij} which is still

orthogonal to w_i but shares the same intersection with the plane defined by w_j along the same edge. The conditions that must be satisfied are:

$$w_{ij} \cdot w_j = 0 \tag{5.12}$$

$$w_{ij} \perp \cap(w_i, w_j) \tag{5.13}$$

$$w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = 0 \tag{5.14}$$

This third condition is the requirement that w_{ij} is orthogonal to all vectors in the intersection of planes i and j.

We'll attempt to solve these equations by using a variable a to account for the discrepency needed to solve these equations.

$$w_{ij} = w_j + a = e_j \cos\theta_j - e_1 \sin\theta_j + a \tag{5.15}$$

$$a = a_1 e_1 + a_2 e_2 + \cdots a_n e_n \tag{5.16}$$

$$w_{ij} \cdot w_i = \sin\theta_j \sin\theta_i - a_1 \sin\theta_j + a_i \cos\theta_i = 0 \tag{5.17}$$

$$w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = a_1 - a_i \tan\theta_i + a_j \tan\theta_j \tag{5.18}$$

Letting $a_1 \neq 0$ requires that $a = e_1 \sin\theta_j - e_j \cos\theta_j$ which is a degenerate case, so let $a_1 = 0$. For every a_k not involved in the intersection of planes, they will have zero

dot product, so we can let them all be 0. Then

$$-a_i \cos\theta_i = \sin\theta_j \sin\theta_i \tag{5.19}$$

$$a_i = -\sin\theta_j \tan\theta_i \tag{5.20}$$

$$a_j \tan\theta_j = \sin\theta_j \tan^2\theta_i \tag{5.21}$$

$$a_j = \cos\theta_j \tan^2\theta_i \tag{5.22}$$

$$w_{ij} = w_j - e_i \sin\theta_j \tan\theta_i + e_j \cos\theta_j \tan^2\theta_i \tag{5.23}$$

Projecting once more onto w_k which also shares an intersection with w_i and w_j, we gain one parameter and one degree of freedom.

$$w_{ijk} = w_k + a = e_k \cos\theta_k - e_1 \sin\theta_k + a \tag{5.24}$$

$$a = a_1 e_1 + a_2 e_2 + \cdots a_n e_n \tag{5.25}$$

$$w_{ijk} \cdot w_i = \sin\theta_k \sin\theta_i - a_1 \sin\theta_k + a_i \cos\theta_i = 0 \tag{5.26}$$

$$w_{ijk} \cdot w_j = \sin\theta_k \sin\theta_j - a_1 \sin\theta_j + a_j \cos\theta_j = 0 \tag{5.27}$$

$$w_{ijk} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = a_1 + a_i \tan\theta_i + a_j \tan\theta_j + a_k \tan\theta_k \tag{5.28}$$

Now, again letting $a_1 = 0$ to avoid a degenerate case and letting all other components of a be zero, we have:

$$-a_i \cos\theta_i = \sin\theta_k \sin\theta_i \tag{5.29}$$

$$a_i = -\sin\theta_k \tan\theta_i \tag{5.30}$$

$$-a_j \cos\theta_j = \sin\theta_k \sin\theta_j \tag{5.31}$$

$$a_j = -\sin\theta_k \tan\theta_j \tag{5.32}$$

$$a_k \tan\theta_k = \sin\theta_k \tan^2\theta_i + \sin\theta_k \tan^2\theta_j \tag{5.33}$$

$$a_k = \cos\theta_k(\tan^2\theta_i + \tan^2\theta_j) \tag{5.34}$$

$$w_{ij} = w_j - e_i \sin\theta_k \tan\theta_i \tag{5.35}$$

$$- e_j \sin\theta_k \tan\theta_j \tag{5.36}$$

$$+ e_k \cos\theta_k(\tan^2\theta_i + \tan^2\theta_j) \tag{5.37}$$

We see now that we can project an arbitrary sequence of such intersecting surfaces with each $w_{i\ldots k}$ with $a_i = -\sin\theta_k \tan\theta_i$ for i less than k and $a_k = \cos\theta_k(\sum_{i=1}^{k} \tan^2\theta_i)$. Now for our projection algorithm, we first determine if all planes are satisfied ($x \cdot v_i \geq 0$ and $x \cdot w_i \leq 0$ for every i.) If they are, then we will project onto each plane and pick the nearest point. If they are not all satisfied, now we'll check each pair of planes w_i and v_i. If both are satisfied, we will skip them, if both are violated, we will set the coordinate in their native direction e_i to 0. When only one of the two planes for all compute all planes that are violated by the position of each query point. For each pair of related planes w_i and v_i, there are three cases. Both satisfied, Both violated, or either or. In the case that both are violated, i.e. $x \cdot v_i < 0$, then $x_1 < 0$, which is to say the nearest point within the wedge in these coordinates will be on the origin.

When both conditions are satisfied, we will ignore this plane except

Now, performing orthogonal projection, we can orthogonally project in sequence for all half-planes not containing x except for v_1 since e_1 is not orthogonal to any of the wedge planes. Once we have projected onto all other violated planes, if the projection onto e_1 is negative, this will mean that the nearest point to x is the origin. For $i \neq 1$, all these projections are of the first form. $x - \sum_i x \cdot e_i$. The first projection of the second form can be done by simple orthogonal projection, however in order to perform the next projection, we must project onto the intersection of the two half-planes without leaving the first plane. We'll need a new plane which is orthogonal to the first plane, but has the same intersection with the second plane. We can compute what is missing to make this projection orthogonal. We have

$$w_i \cdot w_j = \langle e_i \sin(\theta_i) - e_1 \cos(\theta_i), e_j \sin(\theta_j) - e_1 \cos(\theta_j) \rangle \tag{5.38}$$

$$= e_i e_j \sin(\theta_i) \sin(\theta_j) - e_1 e_j \sin(\theta_j) \cos(\theta_j) - e_1 e_i \sin(\theta_i) \cos(\theta_j) + e_1^2 \cos(\theta_j) \cos(\theta_i) \tag{5.39}$$

$$= e_1 \cos(\theta_j) \cos(\theta_i) \tag{5.40}$$

$$\tag{5.41}$$

We can solve for a perturbation in the e_1 direction which will make these two normal vectors orthogonal but due to preserving

5.5 exploring boundary curvature with Random Walks

To analyze decision boundary curvature, we will project samples of points onto the decision boundary and then use Singular Value Decomposition to analyze the *projected points*. In general, this process will involve first selecting two images x_1 and x_2 for which $C(x_1) \neq C(x_2)$. A point x_b for which $C(x_b)$ is ambiguous between $C(x_1)$ and $C(x_2)$. The resulting sample X will be projected to the decision boundary by either computing a loss function that is minimized when each of the classes $C(X)$ flip from $C(x_1)$ to $C(x_2)$ or vice versa respectively and performing gradient descent, or by interpolating to the parent point of opposite class (so for $x \in X$, if $C(x) = C(x_1)$, we will interpolate from x to x_2).

Once a projection has been found, we will take the singular value decomposition (SVD) of this sample and examine it for degeneracy.

Initial comparison of SVDs of decision boundary points yields a single dominant singular value which corresponds with the content of the original image on the boundary. All random noise appears to be mostly orthogonal to this.

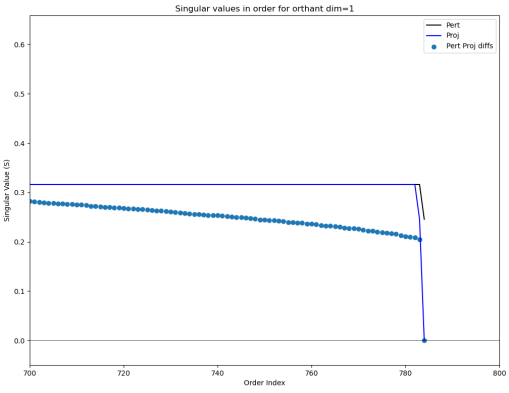
Once this vector is removed, the remaining signal is simply the adversarial attack information.

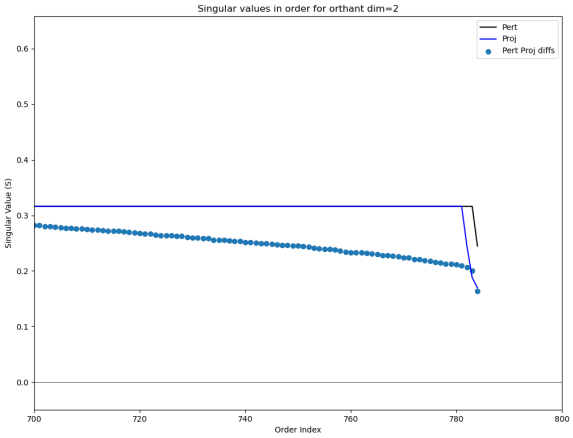
To get a set of singular vectors which to not emphasize the image content, we take random differences among the sampled images and take the SVD of those random differences.

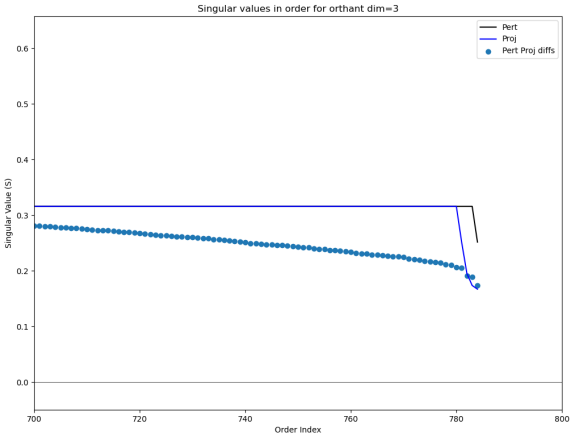
In these first 10 images, this procedure is carried out on the orthant, where samples are generated with mean 0 and are projected onto orthants with increasing numbers

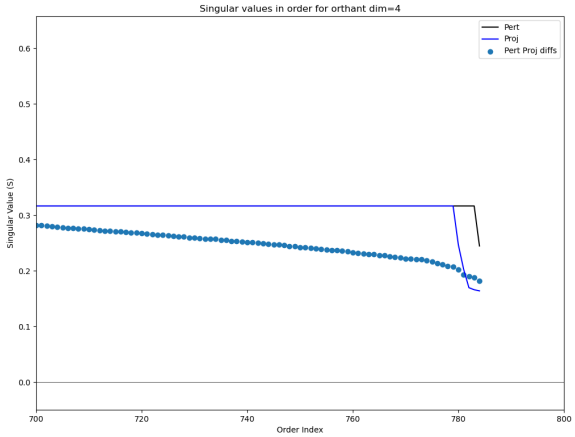
of dimensions. We can see a very clear dropped set of singular values smaller than the rest, which indicate the number of degenerate dimensions. In each of these cases, the dropped singular values match the dimension of the orthant.

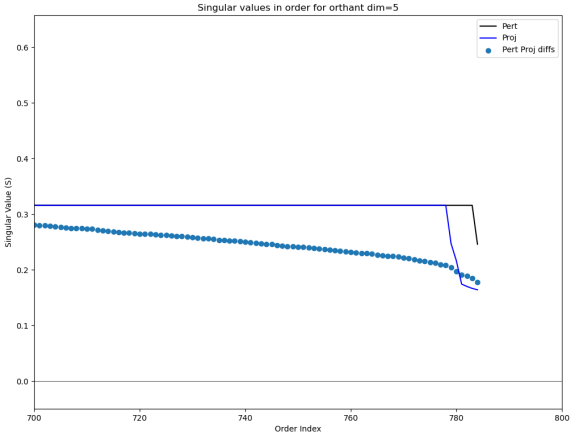
Experiment : A valid experiment here is to measure the difference between the images and the projection to get – in this case – normal vectors to the orthant for each image sampled. The SVD of these normals can be computed to determine the number of dimensions in the projecetion operation. This same procedure can be caried out later in the real practical image projections, although in that case orthogonality is not guaranteed.

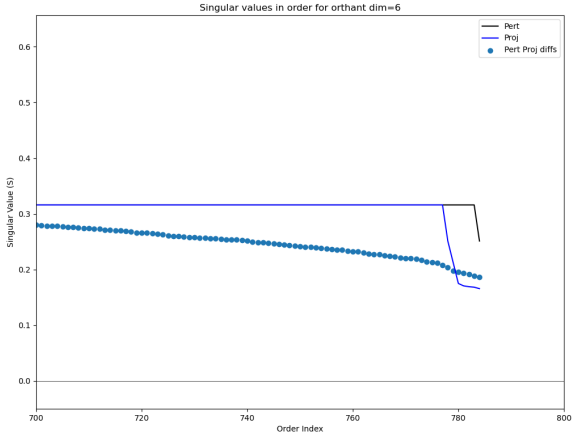


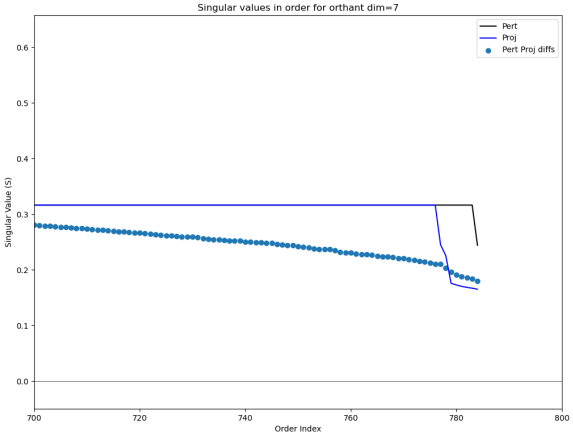


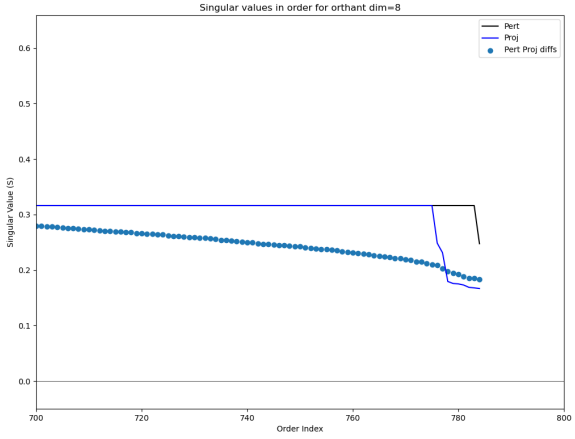

Singular values in order for orthant dim=3
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index

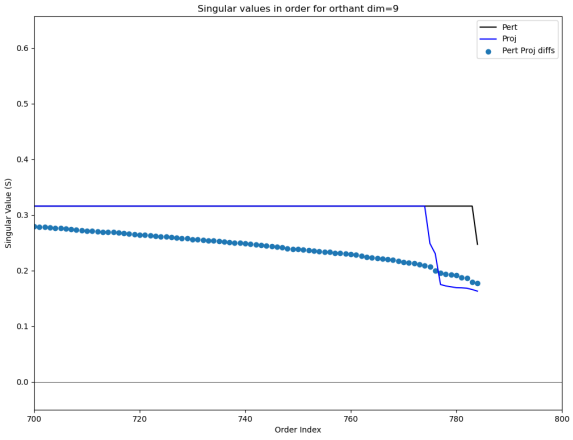

Singular values in order for orthant dim=4
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index

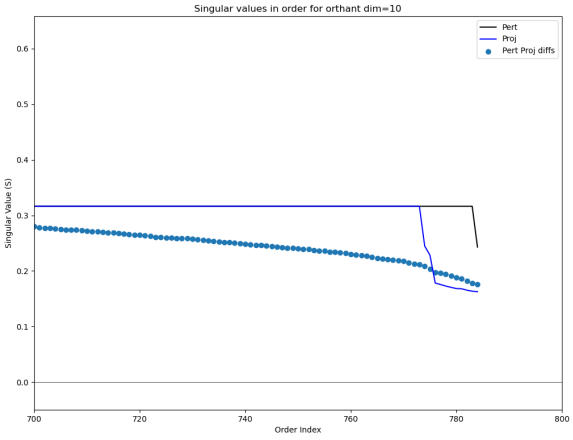

Singular values in order for orthant dim=5
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index


Singular values in order for orthant dim=6
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index


Singular values in order for orthant dim=7
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index


Singular values in order for orthant dim=8
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index

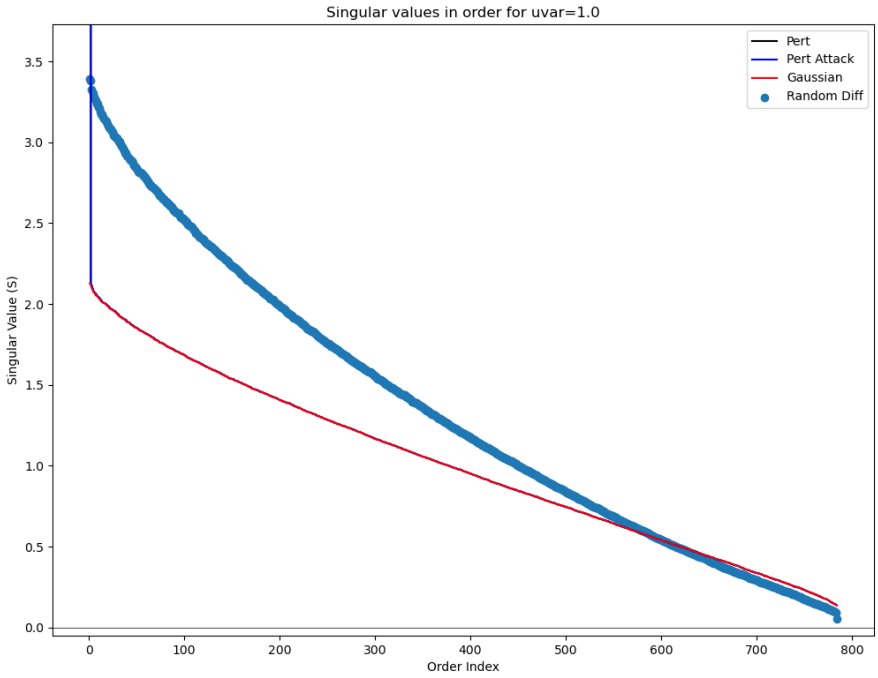




There is a question of how best to measure "normal" vectors for each projection. The most direct computation is to take a sample with small variance around each point on the decision boundary and solve least squares of each of these samples. We have previously observed that *most* of the decision boundary is locally planar.

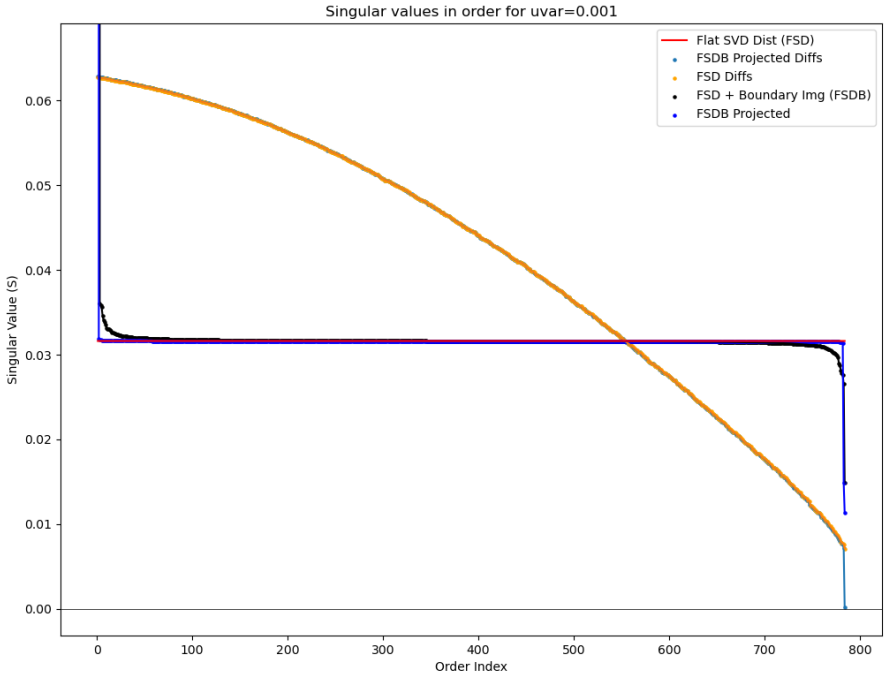
It remains to compare these computed normals with other known quantities, e.g. the gradient computed with respect to adversarial loss functions and the difference between each sampled image and its resultant projection. In addition, it remains to compare the normals of multiple nearby images to determine at what radius of sample the decision boundary begins to show curvature.

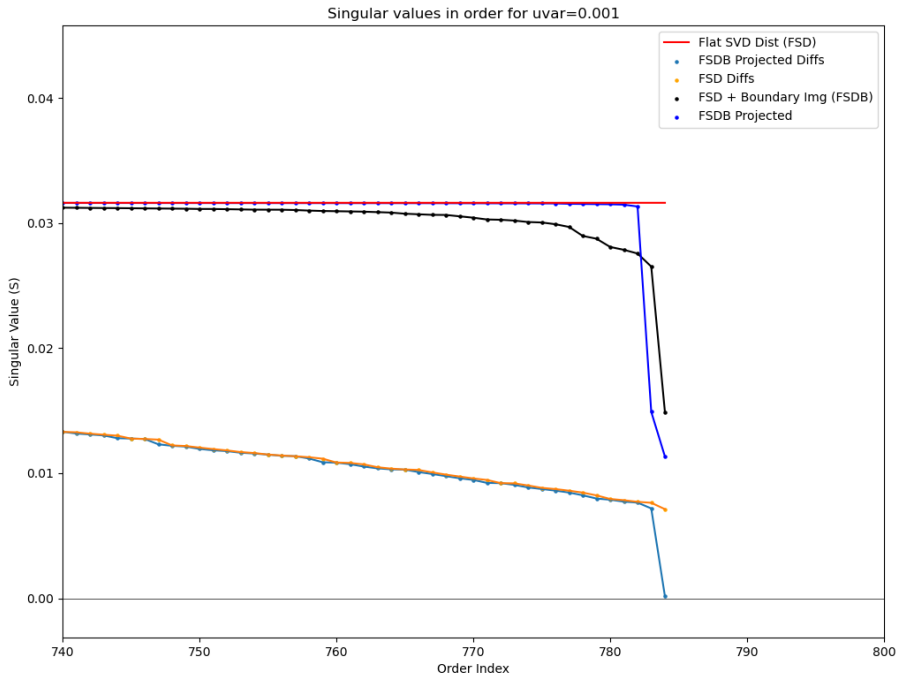
The following image shows singular values in order for a sample around a decision boundary image in MNIST generated by interpolating from a test image to an adversarial image generated from it. This plot is dominated by the natural distribution of singular values for a gaussian and also by the original image information which can be seen as a huge singular value on the far left. We will address both of these factors in following plots.



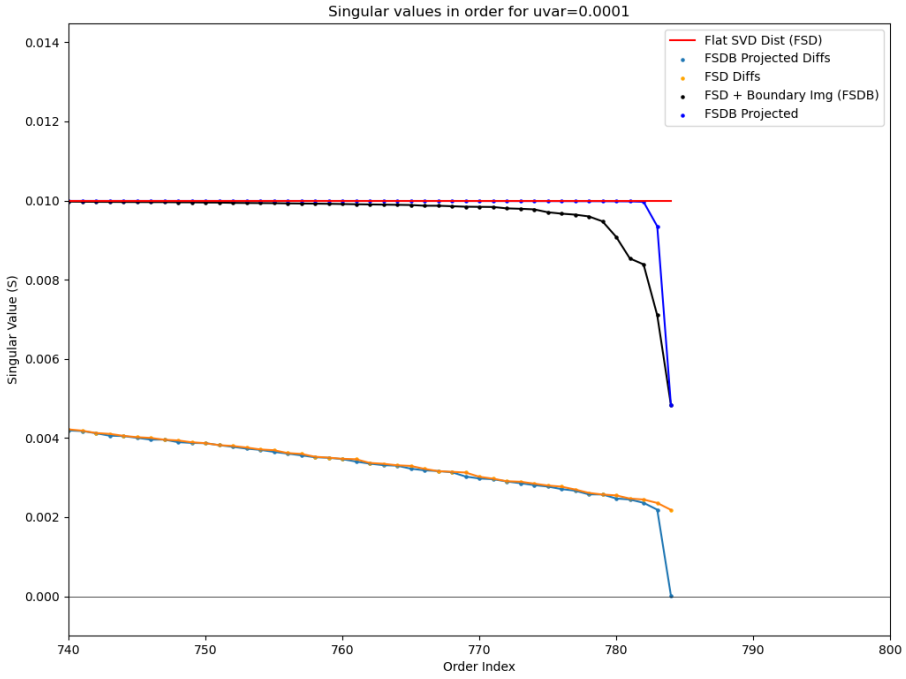
The following plots are generated with a new distribution to replace the gaussian. This "Flat SVD" distribution is generated by first taking a gaussian sample representable by a matrix X, then taking the SVD of this gaussian sample $U\Sigma V = X$, replacing Σ by $|\Sigma|_2 I$. This distribution has the property that its SVD is flat, while maintaining the Frobenius norm of X. This helps to highlight degeneracy in singular values.

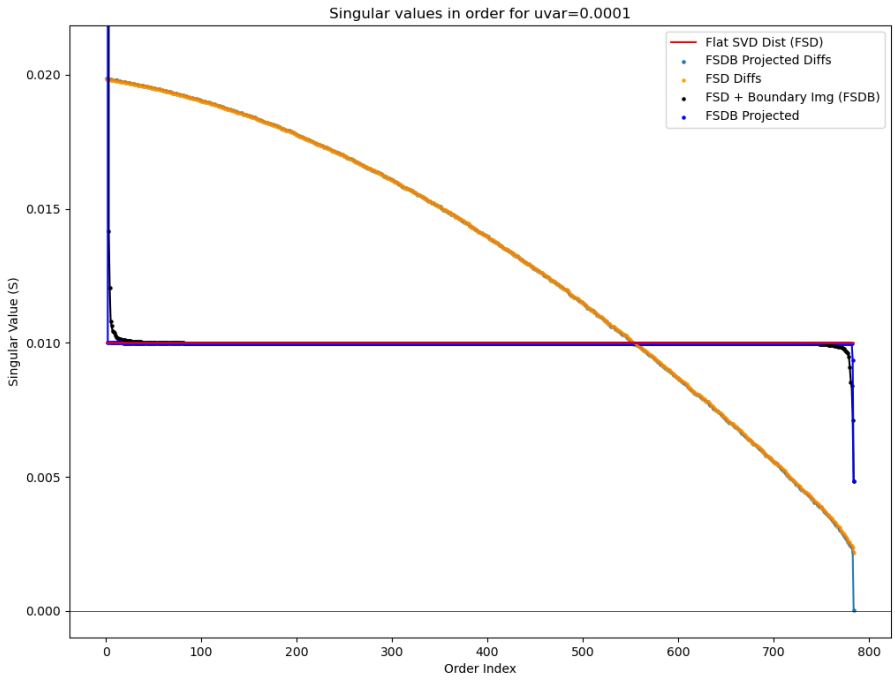
These first two plots are generated by sampling around a decision boundary image found by interpolating between two test images (rather than a test image and an adversarial example generated from it). We note that the SVD contains two degernate values. These seem to correpond with the two original images of the interpolation.


Singular values in order for uvar=0.001
Flat SVD Dist (FSD)
FSDB Projected Diffs
FSD Diffs
FSD + Boundary Img (FSDB)
FSDB Projected
Singular Value (S)
Order Index
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0
100
200
300
400
500
600
700
800



These last two images are generated with the flattened distribution

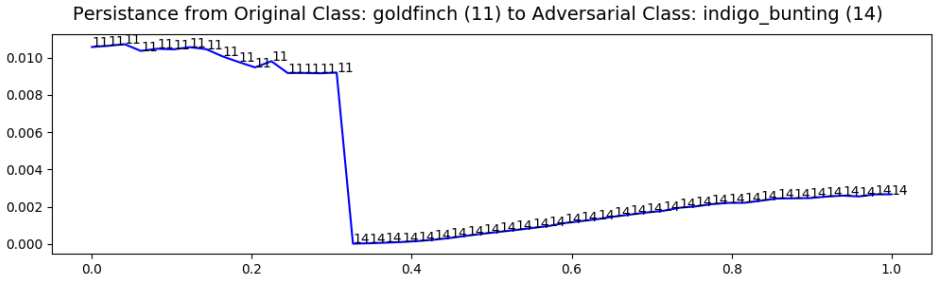

Singular values in order for uvar=0.0001
Flat SVD Dist (FSD)
FSDB Projected Diffs
FSD Diffs
FSD + Boundary Img (FSDB)
FSDB Projected
Singular Value (S)
Order Index
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
740
750
760
770
780
790
800



What these experiments tell us is that most of the time when we cross a decision oundary in the MNIST Dataset, we are crossing at a plane, however if we examine a slightly larger ball around a given point, it will intersect the decision boundary at many other places.

5.6 Re-Examining Persistence

A movivating picture in this research is an image generated while interpolating persistence in the beginning of this research.



The next objective is to examine this particular case with our random walk and projection Tools.

We also wish to augment these tools to include

5.6.1 in probability space

5.6.2 in image space

5.7 define orthants

5.8 skewness

5.9 sampling decision boundaries and analyzing dimensionality with PCA

5.10 neural network attack gradients versus decision boundaries

5.11 skewed orthant recreates persistence picture (?).

5.12 measuring skewness of ANNs

5.13 Relate skewness with dimpled manifold and features not bugs papers

turn observations into are they a definition, a theorem, or a discussion

decision_boundary crossing

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

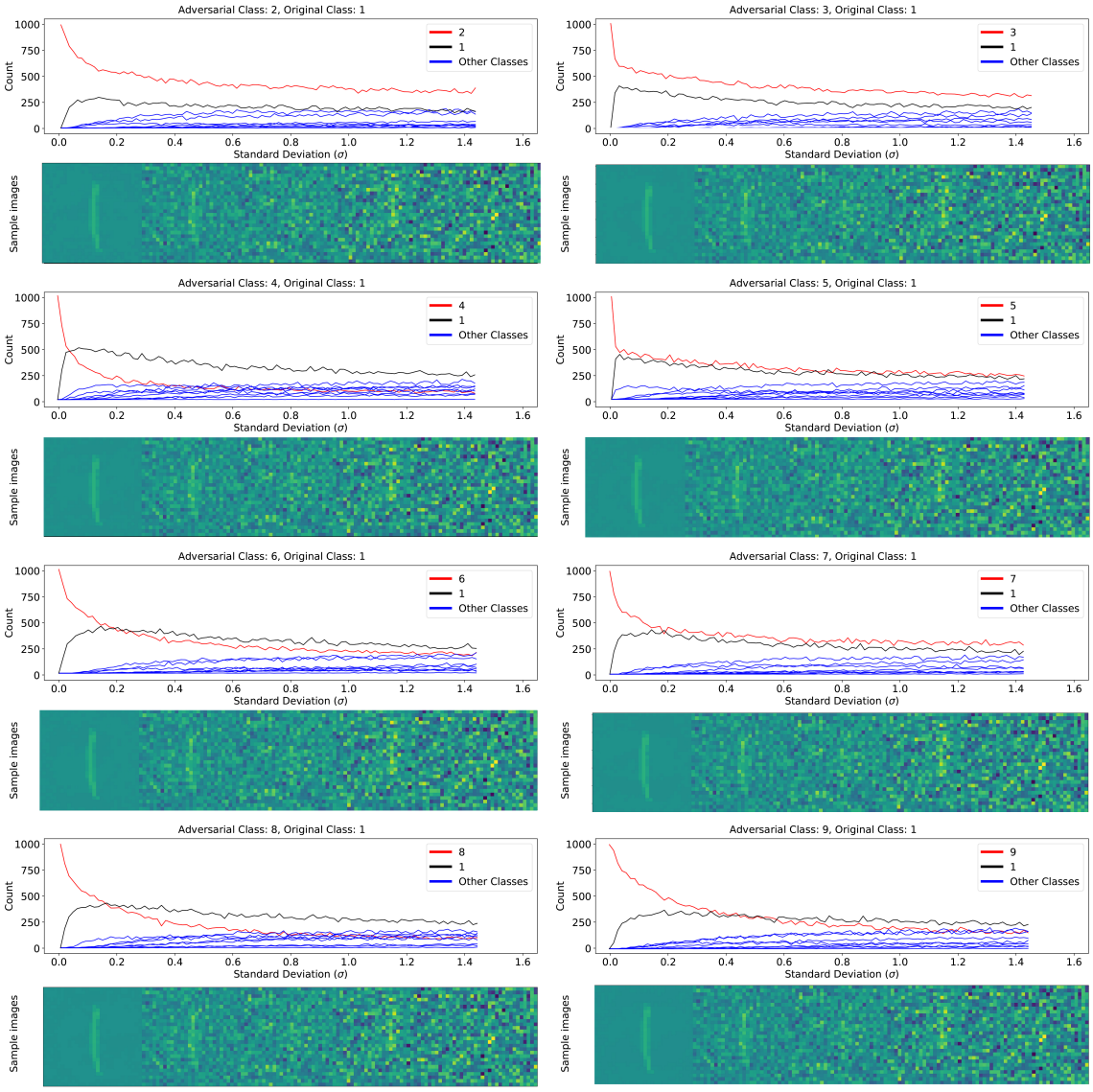


FIGURE B.1: Frequency of each class in Gaussian samples with increasing standard deviations around adversarial attacks of an image of a `1` targeted at classes `2` through `9` on a DNN classifier generated using IGSM. The adversarial class is shown as a red curve. The natural image class (`1`) is shown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

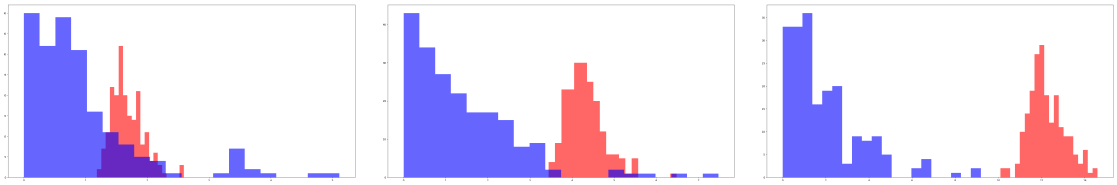


FIGURE B.2: Histograms of 0.7-persistence for FC10-4 (smallest regularization, left), FC10-2 (middle), and FC10-0 (most regularization, right) from Table 3.1. Natural images are in blue, and adversarial images are in red. Note that these are plotted on different scales – higher regularization forces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

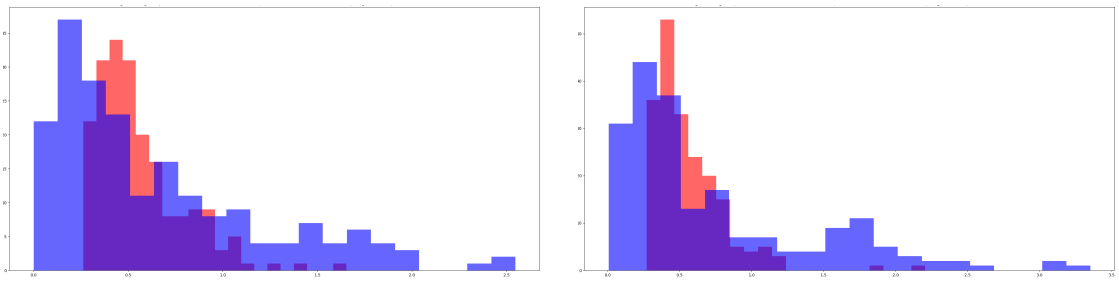


FIGURE B.3: Histograms of 0.7-persistence for FC100-100-10 (left) and FC200-200-10 (right) from Table 3.1. Natural images are in blue, and adversarial images are in red.

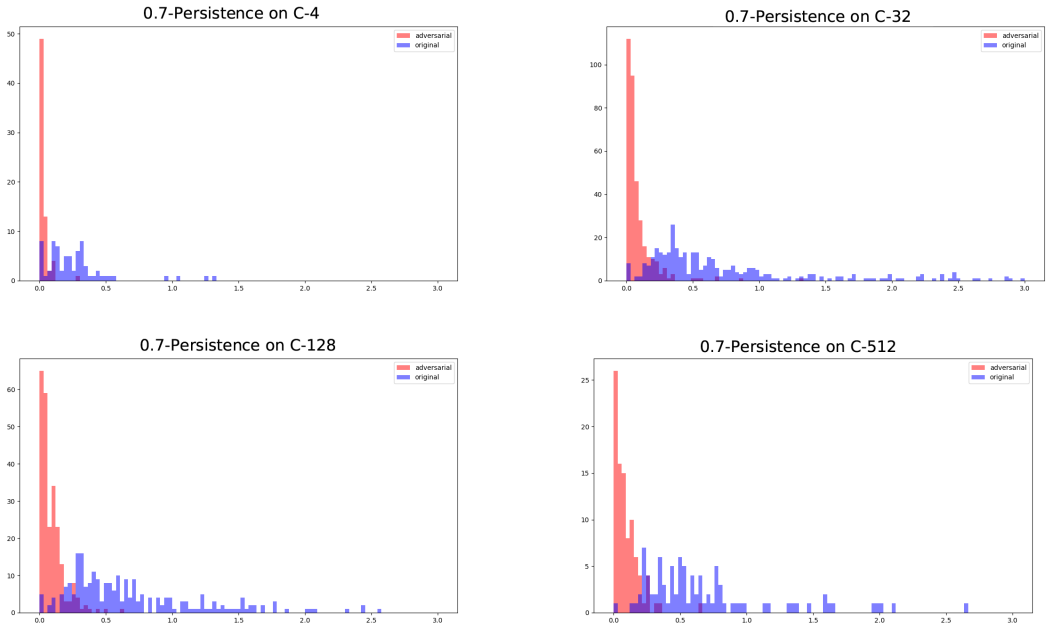


FIGURE B.4: Histograms of 0.7-persistence for C-4 (top left), C-32 (top right), C-128 (bottom left), and C-512 (bottom right) from Table 3.1. Natural images are in blue and adversarial images are in red.

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

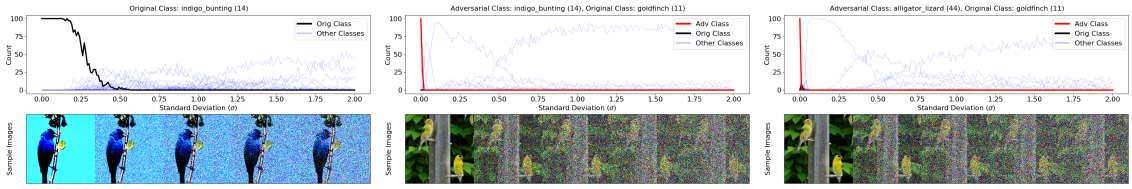


FIGURE B.5: Frequency of each class in Gaussian samples with increasing variance around an `indigo_bunting` image (left), an adversarial example of the image in class `goldfinch` from Figure 3.4 targeted at the `indigo_bunting` class on a alexnet network attacked with PGD (middle), and an adversarial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

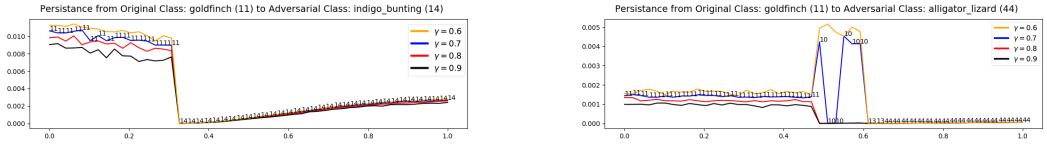


FIGURE B.6: The γ-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) (left) and to an adversarial image generated with PGL in the class `alligator_lizard` (44) (right) on a vgg16 classifier 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).

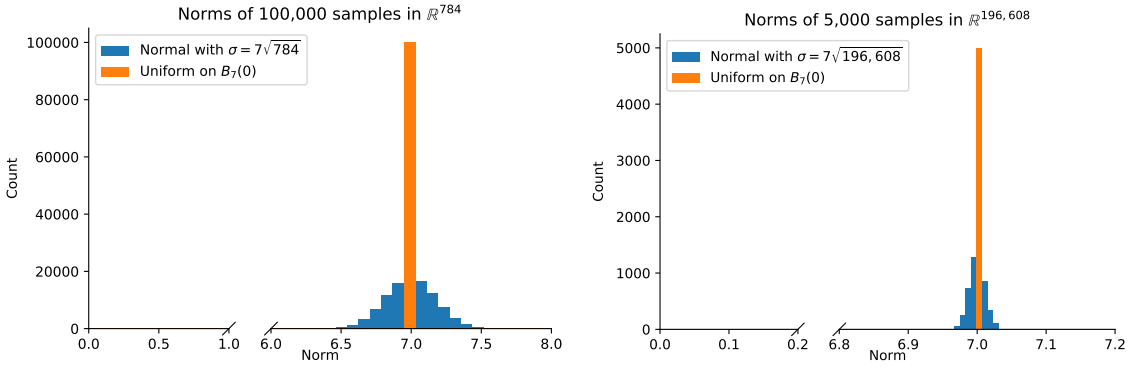


FIGURE B.7: Comparison of the length of samples drawn from $U(B_7(0))$ and $N(0, 7\sqrt{n})$ for $n = 784$, the dimension of MNIST, (left) and $n = 196,608$, the dimension of ImageNet, (right).

Appendix C The EPK is a Kernel

C.1 EPK Proof

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

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

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\int_0^1\langle\nabla_w\hat{y}_{w_s(t,x_i)},\nabla_w\hat{y}_{w_s(t,x_j)}\rangle dt \tag{C.2}$$

$$= \int_0^1\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j\langle\nabla_w\hat{y}_{w_s(t,x_i)},\nabla_w\hat{y}_{w_s(t,x_j)}\rangle dt \tag{C.3}$$

$$= \int_0^1\sum_{i=1}^{n}\sum_{j=1}^{n}\langle\alpha_i\nabla_w\hat{y}_{w_s(t,x_i)},\alpha_j\nabla_w\hat{y}_{w_s(t,x_j)}\rangle dt \tag{C.4}$$

$$= \int_0^1\langle\sum_{i=1}^{n}\alpha_i\nabla_w\hat{y}_{w_s(t,x_i)},\sum_{j=1}^{n}\alpha_j\nabla_w\hat{y}_{w_s(t,x_j)}\rangle dt \tag{C.5}$$

Re-ordering the sums so that their indices match, we have (C.6)

$$= \int_0^1\left\|\sum_{i=1}^{n}\alpha_i\nabla_w\hat{y}_{w_s(t,x_i)}\right\|^2 dt \tag{C.7}$$

$$\geq 0, \tag{C.8}$$

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

□

Remark 3 In the case that our mapping function φ is not symmetric, after re-ordering, we still yield something of the form:

$$= \int_0^1\left\|\sum_{i=1}^{n}\alpha_i\nabla_w\hat{y}_{w_s(t,x_i)}\right\|^2 dt \tag{C.9}$$

(C.10)

The natural asymmetric ϕ is symmetric for every non-training point, so we can partition this sum. For the non-training points, we have symmetry, so for those points we yield exactly the L^2 metric. For the remaining points, if we can pick a Lipschitz constant E along the entire gradient field, then if training steps are enough, then the integral and the discrete step side of the asymmetric kernel will necessarily have positive inner product. In practice, this Lipschitz constant will change during training and for appropriately chosen step size (smaller early in training, larger later in training) we can guarantee positive-definiteness. In particular this only needs to be checked for training points.

C.1.1 The EPK gives an Exact Representation

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:*

$$f_{w_N} = KE(x) := \sum_{s=1}^{N} \sum_{i=1}^{M} a_{i,s} K_{EPK}(x, x', s) + b \tag{4.7}$$

where

$$a_{i,s} = -\varepsilon \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \tag{4.8}$$

$$b = f_{w_0}(x) \tag{4.9}$$

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:

$$\frac{dw_s(t)}{dt} = (w_{s+1} - w_s) \tag{C.11}$$

$$\int \frac{dw_s(t)}{dt} dt = \int (w_{s+1} - w_s) dt \tag{C.12}$$

$$w_s(t) = w_s + t(w_{s+1} - w_s) \tag{C.13}$$

$$\tag{C.14}$$

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

$$\frac{dw_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(X), y_i) = -\varepsilon \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial w_j} \tag{C.15}$$

Applying chain rule and the above substitution, we can write

$$\frac{d\hat{y}}{dt} = \frac{df_{w_s(t)}}{dt} = \sum_{j=1}^{d} \frac{df}{\partial w_j}\frac{\partial w_j}{dt} \tag{C.16}$$

$$= \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j}\left(-\varepsilon \frac{\partial L(f_{w_s(0)}(X_T), Y_T)}{\partial w_j}\right) \tag{C.17}$$

$$= \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j}\left(-\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)}\frac{\partial f_{w_s(0)}(x_i)}{\partial w_j}\right) \tag{C.18}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j}\frac{df_{w_s(0)}(x_i)}{\partial w_j} \tag{C.19}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{C.20}$$

$$\tag{C.21}$$

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

$$y_{s+1} - y_s = \int_0^1 -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) dt \tag{C.22}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt\right) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{C.23}$$

$$\tag{C.24}$$

For all N training steps, we have

$$
\begin{aligned}
y_N &= b + \sum_{s=1}^{N} y_{s+1} - y_s \\
y_N &= b + \sum_{s=1}^{N} -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt \right) \cdot \nabla_w f_{w_s(0)}(x_i) \\
&= b + \sum_{i=1}^{M} \sum_{s=1}^{N} -\varepsilon \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \int_0^1 \left\langle \nabla_w f_{w_s(t,x)}(x), \nabla_w f_{w_s(t,x_i)}(x_i) \right\rangle dt \\
&= b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} \int_0^1 \left\langle \phi_{s,t}(x), \phi_{s,t}(x_i) \right\rangle dt
\end{aligned}
$$

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.

□

C.1.2 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.

$$L(\hat{y}_i, y_i) = \sum_{k=1}^{K} -y_i^k(\hat{y}_i^k) \tag{C.25}$$

$$\tag{C.26}$$

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

$$L(\hat{y}_i, y_i) = -1(\hat{y}_i^k) \tag{C.27}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = -1 \tag{C.28}$$

$$\tag{C.29}$$

If $y_i^k = 0$ then we have

$$L(\hat{y}_i, y_i) = 0(\hat{y}_i^k) \tag{C.30}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = 0 \tag{C.31}$$

$$\tag{C.32}$$

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:

$$y = b - \varepsilon \sum_{i=1}^{N} \sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.33}$$

$$= b - \varepsilon \sum_{i=1}^{N} a_{i,0} \sum_{s=1}^{S} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.34}$$

This formulates a kernel machine where

$$a_{i,0} = \frac{\partial L(f_{w_0}(x_i), y_i)}{\partial f_i} \tag{C.35}$$

$$K(x, x_i) = \sum_{s=1}^{S} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{C.36}$$

$$\phi_{s,t}(x) = \nabla_w f_{w_s(t,x)}(x) \tag{C.37}$$

$$w_s(t, x) = \begin{cases} w_s, x \in X_T \\ \\ w_s(t), x \notin X_T \end{cases} \tag{C.38}$$

$$b = 0 \tag{C.39}$$

□

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.

$$L(f_i, y_i) = (y_i - f_{i,s})^2 \tag{C.40}$$

$$\frac{\partial L(f_i, y_i)}{\partial f_i} = 2(y_i - f_{i,s}) \tag{C.41}$$

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

$$\sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{C.42}$$

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:

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{C.43}$$

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt \tag{C.44}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - f_{i,s} \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt \right) \tag{C.45}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w f_{w_s(t))}(x), f_{i,s} \nabla_w f_{w_s(0)}(x_i) \rangle dt \right) \tag{C.46}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \nabla_w f_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w f_{w_s(t))}(x), \frac{1}{2} \nabla_w (f_{w_s(0)}(x_i))^2 \rangle dt \right) \tag{C.47}$$

$$\tag{C.48}$$

Otherwise, we get the usual

$$= \sum_{s=1}^{S} 2(y_i - f_{i,s}) \int_0^1 \langle \nabla_w f_{w_s(t,x))}(x), \nabla_w f_{w_s(t,x)}(x') \rangle dt \tag{C.49}$$

$$\tag{C.50}$$

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.1.3 Multi-Class Case

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

$$\text{Case 1: } L : \mathbb{R}^K \to \mathbb{R} \tag{C.51}$$

$$\text{Case 2: } L : \mathbb{R}^K \to \mathbb{R}^K \tag{C.52}$$

$$\tag{C.53}$$

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:

$$\frac{df}{dt} = \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \frac{dw_j}{dt} \tag{C.54}$$

$$= -\varepsilon \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(f(x_i), y_i)}{\partial w_j} \tag{C.55}$$

Apply chain rule (C.56)

$$= -\varepsilon \sum_{j=1}^{M} \frac{df(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(f(x_i), y_i)}{\partial f} \frac{df(x_i)}{\partial w_j} \tag{C.57}$$

Let (C.58)

$$A = \frac{df(x)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{C.59}$$

$$B = \frac{dL(f(x_i), y_i)}{df} \in \mathbb{R}^{1 \times K} \tag{C.60}$$

$$C = \frac{df(x_i)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{C.61}$$

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

$$(ABC)^T = (BC)^T A^T = BCA^T \tag{C.62}$$

$$ABC = (BCA^T)^T \tag{C.63}$$

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

$$= -\varepsilon \sum_{j=1}^{M} \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \frac{df(x_i)}{\partial w_j} \left(\frac{df(x)}{\partial w_j} \right)^T \right)^T \tag{C.64}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \sum_{j=1}^{M} \frac{df(x_i)}{\partial w_j} \left(\frac{df(x)}{\partial w_j} \right)^T \right)^T \tag{C.65}$$

$$\tag{C.66}$$

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

$$(\nabla f(x))_{k,j} = \frac{df_k(x)}{\partial w_j} \tag{C.67}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{dL(f(x_i), y_i)}{df} \nabla f(x_i) (\nabla f(x))^T \right)^T \tag{C.68}$$

$$\tag{C.69}$$

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.1.4 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:

$$\frac{dw_s(t)}{dt} = -\varepsilon_s \nabla_w L(f_{w_s(0)}(X), y_i) = -\varepsilon \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(X), y_i)}{\partial w_j} \tag{C.70}$$

This additional dependence of ε on s simply forces us to keep ε inside the summation in equation 4.11.

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

$$\frac{dw_s(t)}{dt} = \varepsilon_{s,l} \nabla_w L(f_{w_s(0)}(x_i), y_i) + \varepsilon_{s-1,l} \nabla_w L(f_{w_{s-1}}(x_i), y_i) + \cdots \tag{C.71}$$

$$= \varepsilon_{s,l} \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial w_j} + \varepsilon_{s-1,l} \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_{s-1}(0)}(x_i), y_i)}{\partial w_j} + \cdots \tag{C.72}$$

This additional dependence of ε on s and l simply results in an additional summation in equation 4.11. 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.1.5 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

$$\bar{\mu}(X') = \left[K(X', X)\right] \left[K(X, X)\right]^{-1} \left[Y\right] \tag{C.73}$$

$$\mathrm{Var}(\bar{\mu}(X') - \mu(X')) = \left[K(X', X')\right] - \left[K(X', X)\right] \left[K(X, X)\right]^{-1} \left[K(X, X')\right] \tag{C.74}$$

$$\tag{C.75}$$

Where

$$[K(A, B)] = \begin{bmatrix} K(A_1, B_1) & K(A_1, B_2) & \cdots \\ K(A_2, B_1) & K(A_2, B_2) & \\ \vdots & & \ddots \end{bmatrix} \tag{C.76}$$

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

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

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

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.

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).

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.

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

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.

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.

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.

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)*.

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.

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

Gilmer, Justin et al. "Adversarial Spheres". In: *6th International Conference on Learning Representations, (ICLR 2018), Vancouver, BC, Canada*.

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).

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-.

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. "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.

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.

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*.

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: *arXiv preprint arXiv:1811.00525*.

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*. Springer, pp. 267–280.

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).

Langley, P. (2000). "Crafting Papers on Machine Learning". In: *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*. Ed. by Pat Langley. Stanford, CA: Morgan Kaufmann, pp. 1207–1216.

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

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.

Ledoux, Michel (1996). "Isoperimetry and Gaussian analysis". In: *Lectures on probability theory and statistics*. Springer, pp. 165–294.

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

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

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.

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.

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).

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

Morvan, Jean-Marie (2008). *Generalized Curvatures*. Springer Publishing Company, Incorporated.

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.

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.

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*.

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.

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.

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: *arXiv preprint arXiv:2106.10151*.

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

Szegedy, Christian et al. (2013). "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199. arXiv: 1312.6199. URL: http://arxiv.org/abs/1312.6199.

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*.

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].

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*.

Xu, Yuancheng et al. (2023). *Exploring and Exploiting Decision Boundary Dynamics for Adversarial Robustness*. arXiv: 2302.03015 [cs.LG].

Yang, Greg et al. (2020). "Randomized smoothing of all shapes and sizes". In: *International Conference on Machine Learning*. PMLR, pp. 10693–10705.