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

Acknowledgements **4**

Quotations **6**

List of Figures **10**

List of Tables **15**

List of Abbreviations **16**

Physical Constants **17**

List of Symbols **18**

Abstract **19**

1 Exact Path Kernels Naturally Decompose Model Predictions **20**

1.1 Introduction 22

1.2 Related Work 24

1.3 Theoretical Justification : Generalized Exact Path Kernel 26

1.4 OOD is enabled by Parameter Gradients 29

1.4.1 Expressing Prior OOD Methods with the gEPK 31

1.4.2 gEPK for OOD 34

1.5 Signal Manifold Dimension Estimated with Training Input Gradients 35

1.6 Conclusion 39

2 Conclusions and Outlook **43**

2.1 Applications to Adversarial Robustness 44

2.2 decision boundary definitions 44

2.3 The Argmax Function 44

2.4 Defining Decision Boundaries 46

2.4.1 Complement Definition 46

2.4.2 Constructive Definition 46

2.4.3 Level Set Definition 46

2.4.4 Images of Decision Boundaries 47

2.5 Properties of Decision Boundaries 48

2.5.1 Delaunay Decision Boundaries 49

2.5.2 Level Set Definition 51

2.5.3 Weighted Delaunay Decision Boundaries 52

2.5.4 Orthant and Wedge 54

2.6 exploring boundary curvature with Random Walks 61

2.7 Re-Examining Persistence 69

2.7.1 in probability space 71

2.7.2 in image space 71

2.8 define orthants 71

2.9 skewness 71

2.10 sampling decision boundaries and analyzing dimensionality with PCA 71

2.11 neural network attack gradients versus decision boundaries 71

2.12 skewed orthant recreates persistence picture (?). 71

2.13 measuring skewness of ANNs 71
2.14 Relate skewness with dimpled manifold and features not bugs papers 71

A Attacks **74**
A.1 L-BFGS 74

B Persistence Tools **76**
B.1 Bracketing Algorithm 76
B.2 Bracketing Algorithm 76
B.3 Convolutional neural networks used 76
B.4 Additional Figures 78
B.4.1 Additional figures from MNIST 78
B.4.2 Additional figures for ImageNet 81
B.5 Concentration of measures 84

C The EPK is a Kernel **86**
C.0.1 When is an Ensemble of Kernel Machines itself a Kernel Machine? 86
C.0.2 Multi-Class Case 90
Case 1 Scalar Loss 90
C.0.3 Schemes Other than Forward Euler (SGD) 93
C.0.4 Variance Estimation 94

Bibliography **96**

List of Figures

1.1 The gEPK naturally provides a measure of input dimension. This plot shows the CDF of the explained variation of training point sensitivities $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Different datasets are color coded to show differences in signal dimension. Decomposing the input space in this way provides a view of the signal dimension around individual test points. For a toy problem (3 Gaussian distributions embedded in 100 dimensional space) the model only observes between 2 and 3 unique variations which contribute to 95% of the information required for prediction. Meanwhile the dimension of the signal manifold observed by the model around MNIST and CIFAR test points is approximately 94 and 1064 respectively. 23

1.2 OOD detection using difference in training vs. test gradients. As the purpose of this paper is not to develop state of the art OOD detection methods, a comparison with recent benchmarks is not provided. Instead, a proof of concept that the gEPK can perform OOD detection is given. Left histogram shows norms of vectors projected onto the gradient weight space defined by the gEPK on MNIST and FMNIST. Right plot shows the number of components required to explain 95% variation in weight space across training for a toy problem (three Gaussian distributions embedded in 100 dimensions). 30

1.3 Explained Variance Ratio of parameter gradients. Left: MNIST, Right: CIFAR. 95% of variation can be explained with a relatively low number of components in both cases. 33

1.4 Left: Visualization of training point input gradients on test points compared between two models. Positive contribution (black) and negative contribution (red) of each training datum to the prediction for each test point. Elements in the grid are $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Right: By taking these individual gradient contributions for a test point and computing the SVD across the training, the significant modes of variation in the input space can be measured (sigma squared). Top is log scale of the full spectrum, bottom shows the first 10 components. Note that this decomposition selects similar, but not identical, modes of variation across test points and even across different models. Components in SVD plots are sorted using Test Point A on Model A. 41

1.5 Differences between observing gradients in input space vs. weight space. Left: CDF of explained variation parameter space. Right: CDF of explained variation input space. Red solid line indicates a model at random initialization while the blue solid line represents the fully trained state. From random initialization, the number of principal components required to achieve 95% explained variation decreases in both cases. Note that at random initialization, the weight space gradients already have only a few directions accounting for significant variation. Disentangling the data dimension using weight space gradients is less effective than doing so in input space (Shamir, Melamed, and BenShmuel, 2021). 42

2.1 The gEPK naturally provides a measure of input dimension. This plot shows the CDF of the explained variation of training point sensitivities $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Different datasets are color coded to show differences in signal dimension. Decomposing the input space in this way provides a view of the signal dimension around individual test points. For a toy problem (3 Gaussian distributions embedded in 100 dimensional space) the model only observes between 2 and 3 unique variations which contribute to 95% of the information required for prediction. Meanwhile the dimension of the signal manifold observed by the model around MNIST and CIFAR test points is approximately 94 and 1064 respectively. 73

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. 79

B.2 Histograms of 0.7-persistence for FC10-4 (smallest regularization, left), FC10-2 (middle), and FC10-0 (most regularization, right) from Table **??**. 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. 80

B.3 Histograms of 0.7-persistence for FC100-100-10 (left) and FC200-200-10 (right) from Table **??**. Natural images are in blue, and adversarial images are in red. 80

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 **??**. Natural images are in blue and adversarial images are in red. 81

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 **??** 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. 82

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. 84

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). 85

List of Tables

B.1 Structure of the CNNs C-Ch used in Table **??** 78

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 Exact Path Kernels Naturally Decompose Model Predictions

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

This paper proposes a generalized exact path kernel gEPK which naturally decomposes model predictions into localized input gradients or parameter gradients. Many cutting edge out-of-distribution (OOD) detection methods are in effect projections onto a reduced representation of the gEPK parameter gradient subspace. This decomposition is also shown to map the significant modes of variation that define how model predictions depend on training input gradients at arbitrary test points. These local features are independent of architecture and can be directly compared between models. Furthermore this method also allows measurement of signal manifold dimension and can inform theoretically principled methods for OOD detection on pre-trained models.

1.1 Introduction

Out-of-distribution (OOD) detection for machine learning models is a new, quickly growing field important to both reliability and robustness (**filos2020**; Hendrycks and Dietterich, 2019; Biggio et al., 2014; Hendrycks and Gimpel, 2017; Silva et al., 2023; Yang et al., 2021). Recent results have empirically shown that parameter gradients are highly informative for OOD detection (**djurisic2023**; Behpour et al., 2023; Huang, Geng, and Li, 2021a). To our knowledge, this paper is the first to present theoretical justifications which explain the surprising effectiveness of parameter gradients for OOD detection.

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

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

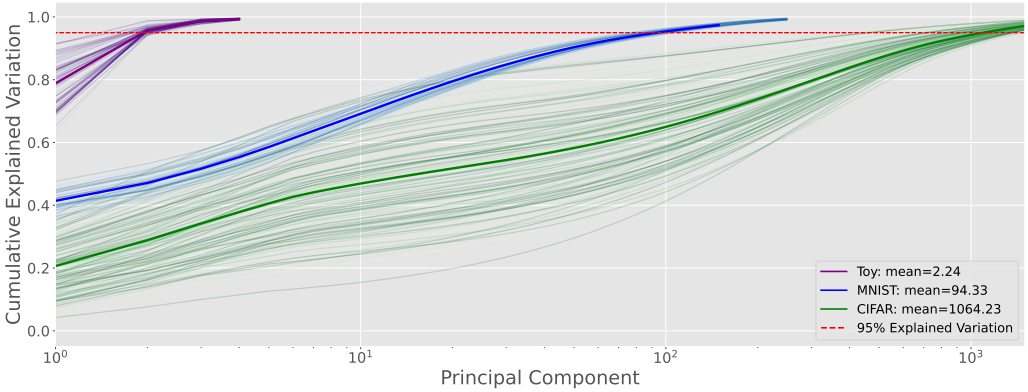


FIGURE 1.1: The gEPK naturally provides a measure of input dimension. This plot shows the CDF of the explained variation of training point sensitivities $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Different datasets are color coded to show differences in signal dimension. Decomposing the input space in this way provides a view of the signal dimension around individual test points. For a toy problem (3 Gaussian distributions embedded in 100 dimensional space) the model only observes between 2 and 3 unique variations which contribute to 95% of the information required for prediction. Meanwhile the dimension of the signal manifold observed by the model around MNIST and CIFAR test points is approximately 94 and 1064 respectively.

Srinivas, Bordt, and Lakkaraju (2023) around a given test point.

In short, this work leverages the gEPK to:

- Introduce a theoretical framework that provides a decomposition in terms of the previously proposed EPK.
- Generalize and explain the success of recent successful methods in OOD.
- Showcase OOD using natural gEPK based decomposition of model predictions in terms of parameter gradients.
- Measure exact input variations and signal manifold dimension around arbitrary test points.

The primary contributions of this paper are theoretical in nature: establishing the exact representation theorem in Section 1.3 and writing several leading OOD detection methods in terms of this representation. The preliminary experimental results also support practical tasks of out-of-distribution (OOD) detection and estimating signal manifold dimension.

1.2 Related Work

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

There are several schools of thought, whether OOD data can be learned (**pillai2013**; Huang and Li, 2021; Mohseni et al., 2020; He et al., 2015; Fumera and Roli, 2002), which part of a model should be interrogated in order to identify OOD examples (Liu et al., 2020; Lin, Roy, and Li, 2021), whether it is a purely statistical question (Lee et al., 2018), or whether it can simply be solved with more data (Chen et al., 2021a; De Silva et al., 2023). The best performing recent approaches have all used relatively simple modifications of model activation or model gradients (**djurisic2023**; Xu et al.,

2023; Sun and Li, 2022; Sun, Guo, and Li, 2021). The first methods we explore (see Section 1.4 relates to the use of model gradients to construct statistics which separate in-distribution (ID) examples from OOD examples. This is fundamentally a geometric approach which should be comparable with the method proposed by Sun et al. (2022) (Gillette and Kur, 2022) The first prominent method of this type was proposed by Liang, Li, and Srikant (2018). ODIN is still a notable method in this space, and has been followed by many more gradient based approaches (Behpour et al., 2023; Huang, Geng, and Li, 2021b) and has caused some confusion about why these methods work so well (Igoe et al., 2022)

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

Khoury and Hadfield-Menell, 2018; Srinivas, Bordt, and Lakkaraju, 2023; Song et al., 2018; Snoek et al., 2019). Along with OOD, uncertainty quantification is increasingly relying on gradients as demonstrated by Lee and AlRegib (2020) and this paper seeks to also support these methods.

1.3 Theoretical Justification : Generalized Exact Path Kernel

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

Theorem 1.3.1 (Generalized Exact Path Kernel (gEPK))**.** *Suppose $f(\cdot;\theta)$ is a differentiable parametric model with parameters $\theta \in \mathbb{R}^M$ and L is a loss function. Furthermore, suppose that f has been trained by a series $\{\theta_s\}_{s=0}^{S}$ of discrete steps composed from a sum of loss gradients for the training set $\sum_i^N \varepsilon \nabla_\theta L(f(x_i), y_i)$ on N training data X_T starting from θ_0. Then for an arbitrary test point x, the trained*

model prediction $f(x;\theta_S)$ can be written:

$$f(x;\theta_S) = f(x;\theta_0) + \sum_{i=1}^{N}\sum_{s=1}^{S} \varepsilon \left(\int_0^1 \varphi_{s,t}(x)dt \right) \frac{dL(f(x_i,\theta_s),y_i)}{d\hat{y}_{\theta_s(0)}} \left(\varphi_{s,0}(x_i)\right) \tag{1.1}$$

$$\varphi_{s,t}(x) \equiv \nabla_\theta f(x;\theta_s(t)), \tag{1.2}$$

$$\theta_s(t) \equiv \theta_s(0) + t(\theta_{s+1}(0) - \theta_s(0)), \ \textit{and} \tag{1.3}$$

$$\hat{y}_{\theta_s(0)} \equiv f(x;\theta_s(0)). \tag{1.4}$$

Proof. Guided by the proof for Theorem 6 from Bell et al. (2023), let θ and $f(\cdot;\theta)$ satisfy the conditions of Theorem 1.3.1, and x be an arbitrary test point. We will measure the change in prediction during one training step from $\hat{y}_s = f(x;\theta_s)$ to $\hat{y}_{s+1} = f(x;\theta_{s+1})$ according to its differential along the interpolation from θ_s to θ_{s+1}. Since we are training using gradient descent, we can write $\theta_{s+1} \equiv \theta_s + \dfrac{d\theta_s(t)}{dt}$. We derive a linear interpolate connecting these states:

$$\frac{d\theta_s(t)}{dt} = (\theta_{s+1} - \theta_s) \tag{1.5}$$

$$\int \frac{d\theta_s(t)}{dt}dt = \int (\theta_{s+1} - \theta_s)dt \tag{1.6}$$

$$\theta_s(t) = \theta_s + t(\theta_{s+1} - \theta_s) \tag{1.7}$$

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

$$\frac{d\theta_s(t)}{dt} = -\varepsilon\nabla_w L(f(X_T;\theta_s(0)),y_i) = -\sum_{i=1}^{N}\varepsilon\sum_{j=1}^{M}\frac{\partial L(f(x_i;\theta_s(0)),y_i)}{\partial\theta^j} \tag{1.8}$$

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

$$\frac{d\hat{y}}{dt} = \frac{df(x;\theta_s(t))}{dt} = \sum_{j=1}^{M} \frac{df}{\partial\theta^j}\frac{\partial\theta^j}{dt} = \sum_{j=1}^{M} \frac{df(x;\theta_s(t))}{\partial\theta^j}\left(-\varepsilon\frac{\partial L(f(X_T,\theta_s(0)),Y_T)}{\partial\theta^j}\right) \tag{1.9}$$

$$= \sum_{j=1}^{M} \frac{df(x;\theta_s(t))}{\partial\theta^j}\left(-\sum_{i=1}^{N}\varepsilon\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\frac{\partial f(x_i;\theta_s(0))}{\partial\theta^j}\right) \tag{1.10}$$

$$= -\sum_{i=1}^{N}\varepsilon\nabla_w f(x;\theta_s(t))\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{1.11}$$

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

$$y_{s+1} - y_s = \int_0^1 -\sum_{i=1}^{N}\varepsilon\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\nabla_w f(x;\theta_s(t))\cdot\nabla_w f(x_i;\theta_s(0))dt \tag{1.12}$$

$$= -\sum_{i=1}^{N}\varepsilon\left(\int_0^1\nabla_w f(x;\theta_s(t))dt\right)\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{1.13}$$

For all N training steps, we have

$$y_N = f(x;\theta_0) + \sum_{s=1}^{N} y_{s+1} - y_s \tag{1.14}$$

$$= f(x;\theta_0) - \sum_{s=1}^{N}\sum_{i=1}^{N}\varepsilon\left(\int_0^1\nabla_w f(x;\theta_s(t))dt\right)\frac{dL(f(x_i;\theta_s(0)),y_i)}{df(x_i;\theta_s(0))}\cdot\nabla_w f(x_i;\theta_s(0)) \tag{1.15}$$

□

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

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

1.4 OOD is enabled by Parameter Gradients

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

$$\left\{\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i); i \in \{1, ..., N\}, s \in \{1, ..., S\}\right\} \tag{1.16}$$

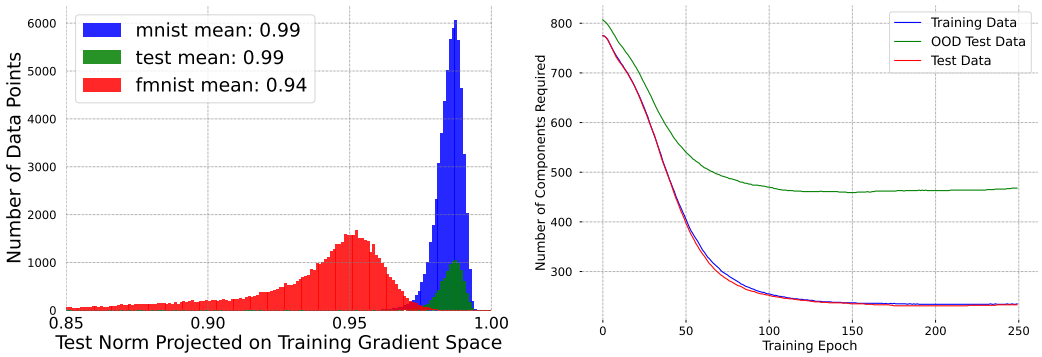


FIGURE 1.2: OOD detection using difference in training vs. test gradients. As the purpose of this paper is not to develop state of the art OOD detection methods, a comparison with recent benchmarks is not provided. Instead, a proof of concept that the gEPK can perform OOD detection is given. Left histogram shows norms of vectors projected onto the gradient weight space defined by the gEPK on MNIST and FMNIST. Right plot shows the number of components required to explain 95% variation in weight space across training for a toy problem (three Gaussian distributions embedded in 100 dimensions).

spans the subspace in which all model predictions can vary. Empirically, we compute the SVD of these training parameter gradients summing over classes in Fig. 1.3. This shows that the space does most of its variation is within only a few dimensions for each test point. Additionally, this corresponds with the well-established notion that the latent dimension of data is much smaller than its ambient dimension. It is by analyzing spectra of this subspace that we may accomplish out-of-distribution (OOD) detection, and, in fact, we will demonstrate that this is how most cutting-edge OOD methods discriminate in practice.

1.4.1 Expressing Prior OOD Methods with the gEPK

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

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

$$\frac{1}{C}\sum_{i}^{C}\frac{\partial L_{CE}(f(x;\theta),i)}{\partial \hat{y}}\nabla_{\theta}f(x;\theta) \tag{1.17}$$

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

also be equivalent through the connection we establish below in Section 1.1 between this decomposition and input gradients which may relate with mapping data manifolds in the Voronoi/Delaunay (Gillette and Kur, 2022) sense.

ReAct, DICE, ASH, and VRA Along with other recent work (Sun, Guo, and Li, 2021; Sun and Li, 2022; Xu et al., 2023), some of the cutting edge for OOD as of early 2023 involves activation truncation techniques like that neatly described by **djurisic2023**. Given a model, $f(x;\theta) = f^{\text{extract}}(\cdot;\theta_{\text{extract}}) \circ f^{\text{represent}}(\cdot;\theta_{\text{represent}}) \circ f^{\text{classify}}(\cdot;\theta_{\text{classify}})$, and an input, x, a prediction, $f(x,\theta)$, is computed forward through the network. This yields a vector of activations, $A(x,\theta_{\text{represent}})$, in the representation layer of the network. This representation is then pruned down to the p^{th} percentile by setting any activations below that percentile to zero. **djurisic2023** mention that ASH does not depend on statistics from the training data, however by chain rule, high activations will correspond with high parameter gradients. That means that this truncation is picking a representation for which $\left\langle \nabla_\theta f(x,\theta_{\text{represent}}), \frac{dL(\hat{y}(x_i),y_i)}{d\hat{y}} \nabla_\theta f(x_i,\theta_{\text{represent}}) \right\rangle$ is high for many training data, x_i. This is effectively a projection onto the parameter tangent space of the training data with the highest variation. This may explain some part of the performance advantage of these method.

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

$$\nabla_\theta L(x,y) = (\theta x - y)x^T = \Omega x^T \tag{1.18}$$

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

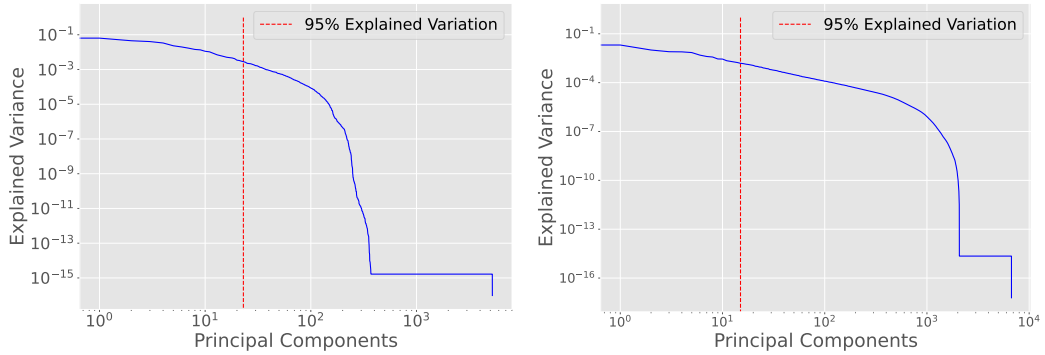


FIGURE 1.3: Explained Variance Ratio of parameter gradients. Left: MNIST, Right: CIFAR. 95% of variation can be explained with a relatively low number of components in both cases.

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

$$\left\| U^L_{ID}\Sigma^L_{ID,k}(V^L_{ID})^T \right\|^2_F \geq \epsilon_{\text{th}} \|R^L_I D\|^2_F. \tag{1.19}$$

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

$$O(x) = (\nabla_{\theta_L} \mathcal{L}(f(x, \theta_L), y)) S^L (S^L)^T \tag{1.20}$$

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

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

1.4.2 gEPK for OOD

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

$$\text{span}\left(\left\{\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i) : x_i \in X_T, 0 \leq s \leq S\right\}\right). \tag{1.21}$$

We can see that most, if not all, of the above methods can be represented by some set of averaging assumptions on the representation provided by the gEPK. We test the ability of the gEPK to perform OOD detection by direct projection onto the full parameter gradient space spanned by its training parameter gradients using a sum over the class outputs in Fig. 1.2. Indeed, the gEPK helps explain the high performance of gradient based methods due to the implicit inclusion of the training parameter space in model predictions. This serves to illuminate the otherwise confusing discrepancy raised by Igoe et al. (2022).

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

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

1.5 Signal Manifold Dimension Estimated with Training Input Gradients

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

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

$$f(x, \theta_{\text{trained}}) = f(x, \theta_0(0)) + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) dt \tag{1.22}$$

$$\varphi_{s,t}(x) = \nabla_\theta f(x, \theta_s(t)) \tag{1.23}$$

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

$$\frac{df(x, \theta_{\text{trained}})}{dx_j} = \frac{df(x, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d\left(\varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{1.24}$$

$$= 0 + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{d\left(\frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{1.25}$$

$$= \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) dt \left(\frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \tag{1.26}$$

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

$$G_j = \sum_s \int_0^1 \varphi_{s,t}(x) dt \left(\frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \tag{1.27}$$

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

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

On MNIST, $G(x)$ creates a matrix which is of size 60000×784 (training points $\times$ input dimension). This matrix represents the exact measure of each training points contribution towards a given test prediction. Of note is that in practice this matrix is full rank on the input space as seen in Figure 1.4. This is despite MNIST having significantly less degrees of variation than its total input size (many pixels in input space are always 0). Figure 1.1 demonstrates that accounting for 95% of the variation requires only 94 (12%) of the 784 components on average. Similarly, on CIFAR accounting for 95% of explained variation requires 1064 (34%) of the 3096 components. It is likely that different training techniques will provide significantly different signal manifolds and consequently different numbers of components. Examining these effects may lead to deeper understanding of optimization techniques.

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

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

$$\frac{df(x_j, \theta_{\text{trained}})}{dx_j} = \frac{df(x_j, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d\left(\varphi_{s,t}(x_j) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i)\right)}{dx_j} dt \tag{1.28}$$

The left hand side is computable without path-decomposition and so can be computed for each training datum to create a gradient matrix, $H_{\theta_{\text{trained}}}$. Another term, $\frac{df(x_j, \theta_0(0))}{dx_j}$ is also easily computable, yielding another matrix H_{θ_0}. By comparing the rank and span of $H_{\theta_{\text{trained}}}$ and H_{θ_0} we can understand to what extent the model's spatial representation of the data is due to the initial parameter selection and how much is due to the training path. Also, $H_{\theta_{\text{trained}}}$ provides sample of gradients across all training data, which in some sense must be spanned by the model's implicit subspace basis. Despite missing the granular subspace information, the rank of this gradient matrix and its explained variation computed using SVD should be related to the model's implicit subspace rank. It should be noted that while there is a direct relationship between a models variations in input space and weight space, Figure 1.5 shows that this mapping changes greatly from the beginning to end of training and that this spectrum starts out wide (high dimensional) for θ_0 and much more focused (low dimensional) for θ_T.

One interesting property of using input gradients for training data decomposed according to Eq. 1.27 is the ability to compare input gradients across models with different initial parameters and even different architectures. Figure 1.4 demonstrates that two models with different random initializations which have been trained on

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

1.6 Conclusion

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

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

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

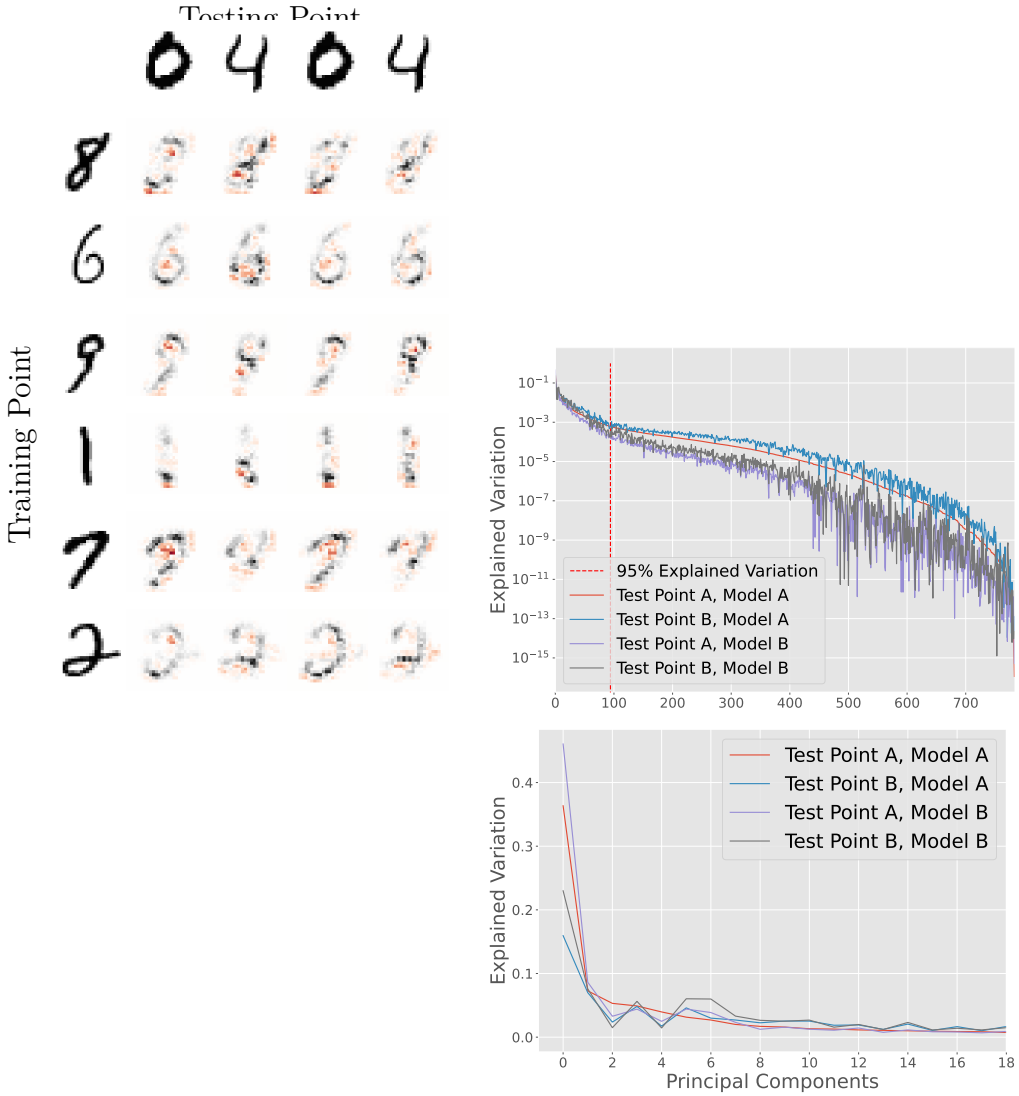


FIGURE 1.4: Left: Visualization of training point input gradients on test points compared between two models. Positive contribution (black) and negative contribution (red) of each training datum to the prediction for each test point. Elements in the grid are $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$. Right: By taking these individual gradient contributions for a test point and computing the SVD across the training, the significant modes of variation in the input space can be measured (sigma squared). Top is log scale of the full spectrum, bottom shows the first 10 components. Note that this decomposition selects similar, but not identical, modes of variation across test points and even across different models. Components in SVD plots are sorted using Test Point A on Model A.

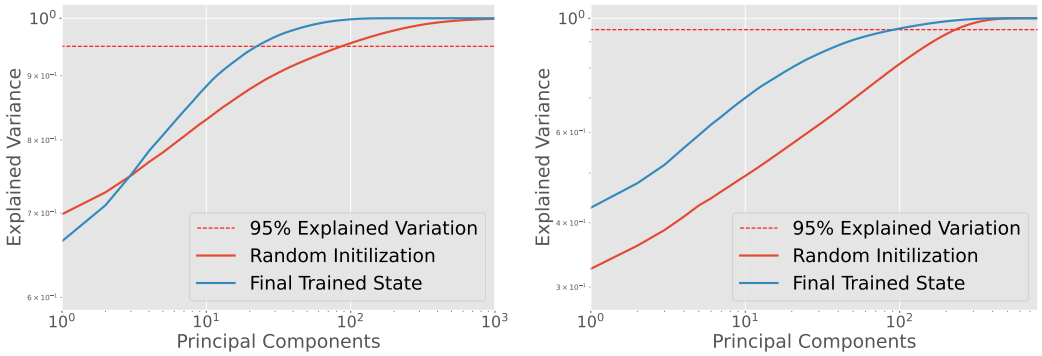


FIGURE 1.5: Differences between observing gradients in input space vs. weight space. Left: CDF of explained variation parameter space. Right: CDF of explained variation input space. Red solid line indicates a model at random initialization while the blue solid line represents the fully trained state. From random initialization, the number of principal components required to achieve 95% explained variation decreases in both cases. Note that at random initialization, the weight space gradients already have only a few directions accounting for significant variation. Disentangling the data dimension using weight space gradients is less effective than doing so in input space (Shamir, Melamed, and BenShmuel, 2021).

Chapter 2 Conclusions and Outlook

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

We can summarize the first 3 chapters as posing some fundamental geometric questions related to machine-learning robustness. Chapters 4 and 5 as propose a new framework for analyzing geometric properties and demonstrating that this framework can be applied very generally. From here, there are several important directions for future research. First is the application of these methods directly to the questions from Chapters 1-3. Second is the complete generalization of this new framework including conditions under which such representations live in Banach spaces or Hilbert spaces and what order of accuracy can be maintained using truncated spectral decompositions. Third is the application of this theory more broadly to the wide variety of spatial problems that are present in machine-learning.

2.1 Applications to Adversarial Robustness

In Fig. 2.1 we can see

2.2 decision boundary definitions

2.3 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:

$$\text{argmax} : \mathbb{R}^k \to \mathcal{P}(C) \tag{2.1}$$

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

2.4 Defining Decision Boundaries

2.4.1 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 $\overline{\{x : x \notin D'_f\}}$.

2.4.2 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 $|f(B_\epsilon(x))| = K$.

2.4.3 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 $a > F(x)_c$.

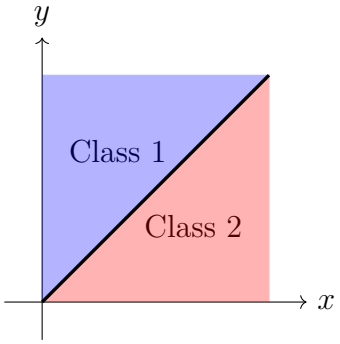
2.4.4 Images of Decision Boundaries

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 $\mathbb{C}$.

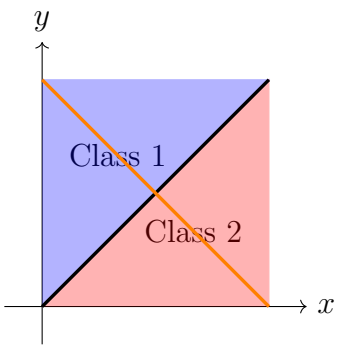
$$X \xrightarrow{F} Y \xrightarrow{\text{argmax}} \mathcal{C} \tag{2.2}$$

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.

2.5 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.

2.5.1 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 2.5.1. *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 2.5.2. *The*

2.5.2 Level Set Definition

Definition 2.5.3. *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 2.5.4. *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.

2.5.3 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 2.5.5. *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 $\{z \in M :$ for all i, we have $w(z)_i \leq a$ 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 2.5.6. *Given a low dimensional manifold M,*

$$\lim_{\varepsilon \to 0} B_\varepsilon(M) = B(M) \tag{2.3}$$

$B(M)$ 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.

2.5.4 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 2.5.7. *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 2.5.8. *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 2.5.9. *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 2.5.10. *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 2.5.11. *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*

$$proj_{\mathbf{v_{i_1}}}(proj_{\mathbf{v_{i_2}}}(proj_{\mathbf{v_{i_3}}}...(\mathbf{x}))) \tag{2.4}$$

(Jordan and Neumann, 1935)

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

$$proj_{\mathbf{p_i}}(\mathbf{x}) = x - x \cdot v_i \tag{2.5}$$

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,

$$w_i \cdot w_j = \langle e_i \cos(\theta_i) - e_1 \sin(\theta_i), e_j \cos(\theta_j) - e_1 \sin(\theta_j) \rangle \tag{2.6}$$

$$= e_i e_j \cos(\theta_i)\cos(\theta_j) - e_1 e_j \cos(\theta_j)\sin(\theta_j) - e_1 e_i \cos(\theta_i)\sin(\theta_j) + e_1^2 \sin(\theta_j)\sin(\theta_i) \tag{2.7}$$

$$= e_1 \sin(\theta_j)\sin(\theta_i) \tag{2.8}$$

$$\tag{2.9}$$

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

$$v_1 \cdot w_i = \langle e_1, -e_1 \sin(\theta_i) + e_i \cos(\theta_i) \rangle \tag{2.10}$$

$$= -\sin(\theta_i) \tag{2.11}$$

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:

$$w_{ij} \cdot w_j = 0 \tag{2.12}$$

$$w_{ij} \perp \cap(w_i, w_j) \tag{2.13}$$

$$w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = 0 \tag{2.14}$$

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.

$$w_{ij} = w_j + a = e_j \cos\theta_j - e_1 \sin\theta_j + a \tag{2.15}$$

$$a = a_1 e_1 + a_2 e_2 + \cdots a_n e_n \tag{2.16}$$

$$w_{ij} \cdot w_i = \sin\theta_j \sin\theta_i - a_1 \sin\theta_j + a_i \cos\theta_i = 0 \tag{2.17}$$

$$w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = a_1 - a_i \tan\theta_i + a_j \tan\theta_j \tag{2.18}$$

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

$$-a_i \cos\theta_i = \sin\theta_j \sin\theta_i \tag{2.19}$$

$$a_i = -\sin\theta_j \tan\theta_i \tag{2.20}$$

$$a_j \tan\theta_j = \sin\theta_j \tan^2\theta_i \tag{2.21}$$

$$a_j = \cos\theta_j \tan^2\theta_i \tag{2.22}$$

$$w_{ij} = w_j - e_i \sin\theta_j \tan\theta_i + e_j \cos\theta_j \tan^2\theta_i \tag{2.23}$$

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.

$$w_{ijk} = w_k + a = e_k \cos\theta_k - e_1 \sin\theta_k + a \tag{2.24}$$

$$a = a_1 e_1 + a_2 e_2 + \cdots a_n e_n \tag{2.25}$$

$$w_{ijk} \cdot w_i = \sin\theta_k \sin\theta_i - a_1 \sin\theta_k + a_i \cos\theta_i = 0 \tag{2.26}$$

$$w_{ijk} \cdot w_j = \sin\theta_k \sin\theta_j - a_1 \sin\theta_j + a_j \cos\theta_j = 0 \tag{2.27}$$

$$w_{ijk} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = a_1 + a_i \tan\theta_i + a_j \tan\theta_j + a_k \tan\theta_k \tag{2.28}$$

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

$$-a_i \cos\theta_i = \sin\theta_k \sin\theta_i \tag{2.29}$$

$$a_i = -\sin\theta_k \tan\theta_i \tag{2.30}$$

$$-a_j \cos\theta_j = \sin\theta_k \sin\theta_j \tag{2.31}$$

$$a_j = -\sin\theta_k \tan\theta_j \tag{2.32}$$

$$a_k \tan\theta_k = \sin\theta_k \tan^2\theta_i + \sin\theta_k \tan^2\theta_j \tag{2.33}$$

$$a_k = \cos\theta_k(\tan^2\theta_i + \tan^2\theta_j) \tag{2.34}$$

$$w_{ij} = w_j - e_i \sin\theta_k \tan\theta_i \tag{2.35}$$

$$- e_j \sin\theta_k \tan\theta_j \tag{2.36}$$

$$+ e_k \cos\theta_k(\tan^2\theta_i + \tan^2\theta_j) \tag{2.37}$$

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

$$w_i \cdot w_j = \langle e_i \sin(\theta_i) - e_1 \cos(\theta_i), e_j \sin(\theta_j) - e_1 \cos(\theta_j) \rangle \tag{2.38}$$

$$= e_i e_j \sin(\theta_i) \sin(\theta_j) - e_1 e_j \sin(\theta_j) \cos(\theta_j) - e_1 e_i \sin(\theta_i) \cos(\theta_j) + e_1^2 \cos(\theta_j) \cos(\theta_i) \tag{2.39}$$

$$= e_1 \cos(\theta_j) \cos(\theta_i) \tag{2.40}$$

$$\tag{2.41}$$

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

2.6 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.

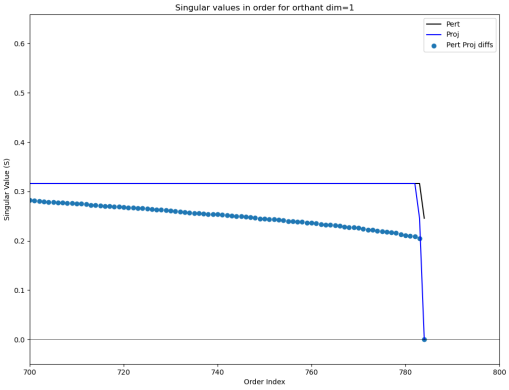
Initial comparison of SVDs of decision boundary points yields a single dominant singular value which corresponds with the content of the original image on the boundary. All random noise appears to be mostly orthogonal to this.

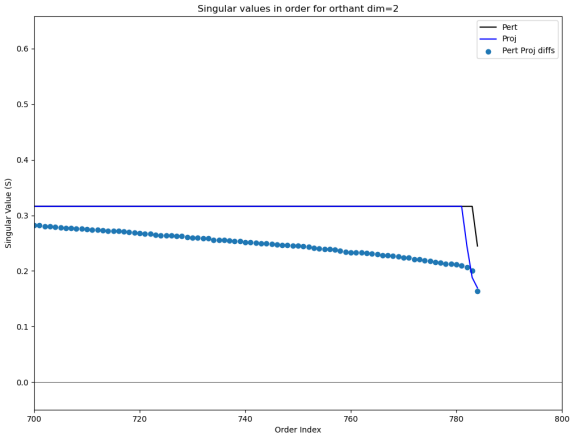
Once this vector is removed, the remaining signal is simply the adversarial attack information.

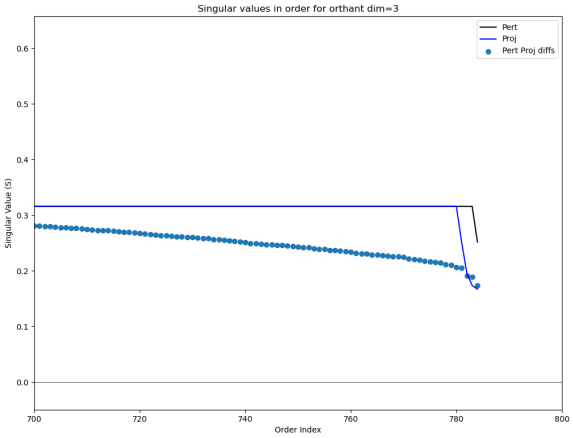
To get a set of singular vectors which to not emphasize the image content, we take random differences among the sampled images and take the SVD of those random differences.

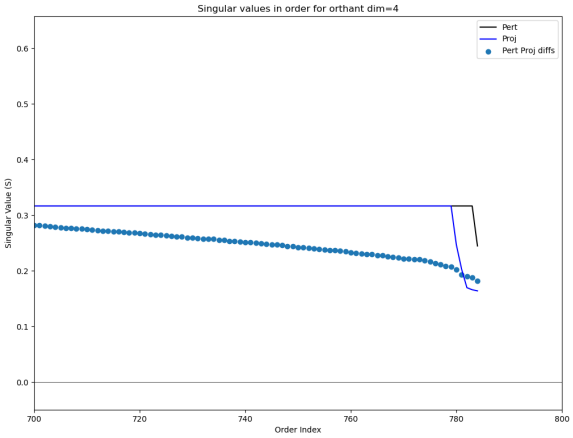
In these first 10 images, this procedure is carried out on the orthant, where samples are generated with mean 0 and are projected onto orthants with increasing numbers of dimensions. We can see a very clear dropped set of singular values smaller than the rest, which indicate the number of degenerate dimensions. In each of these cases, the dropped singular values match the dimension of the orthant.

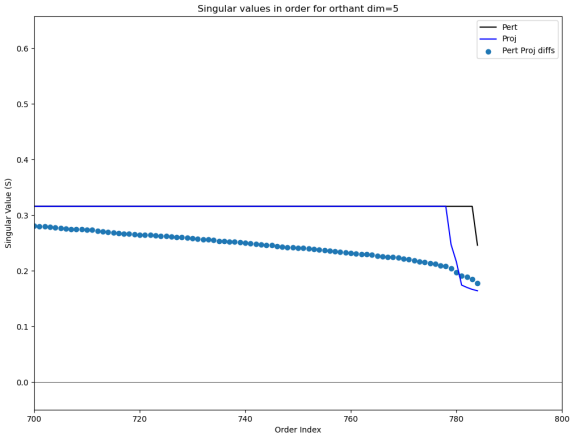
Experiment : A valid experiment here is to measure the difference between the images and the projection to get – in this case – normal vectors to the orthant for each image sampled. The SVD of these normals can be computed to determine the number of dimensions in the projecetion operation. This same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

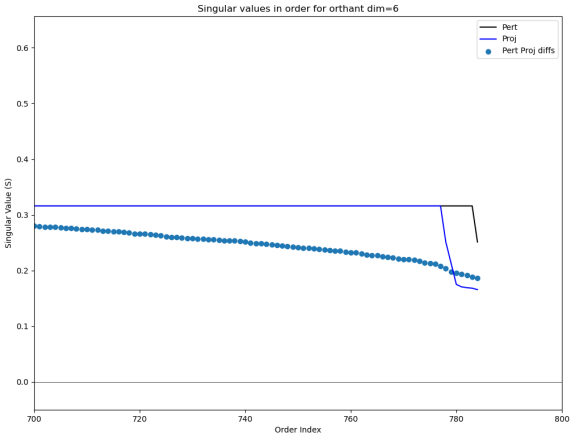

Singular values in order for orthant dim=1
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index

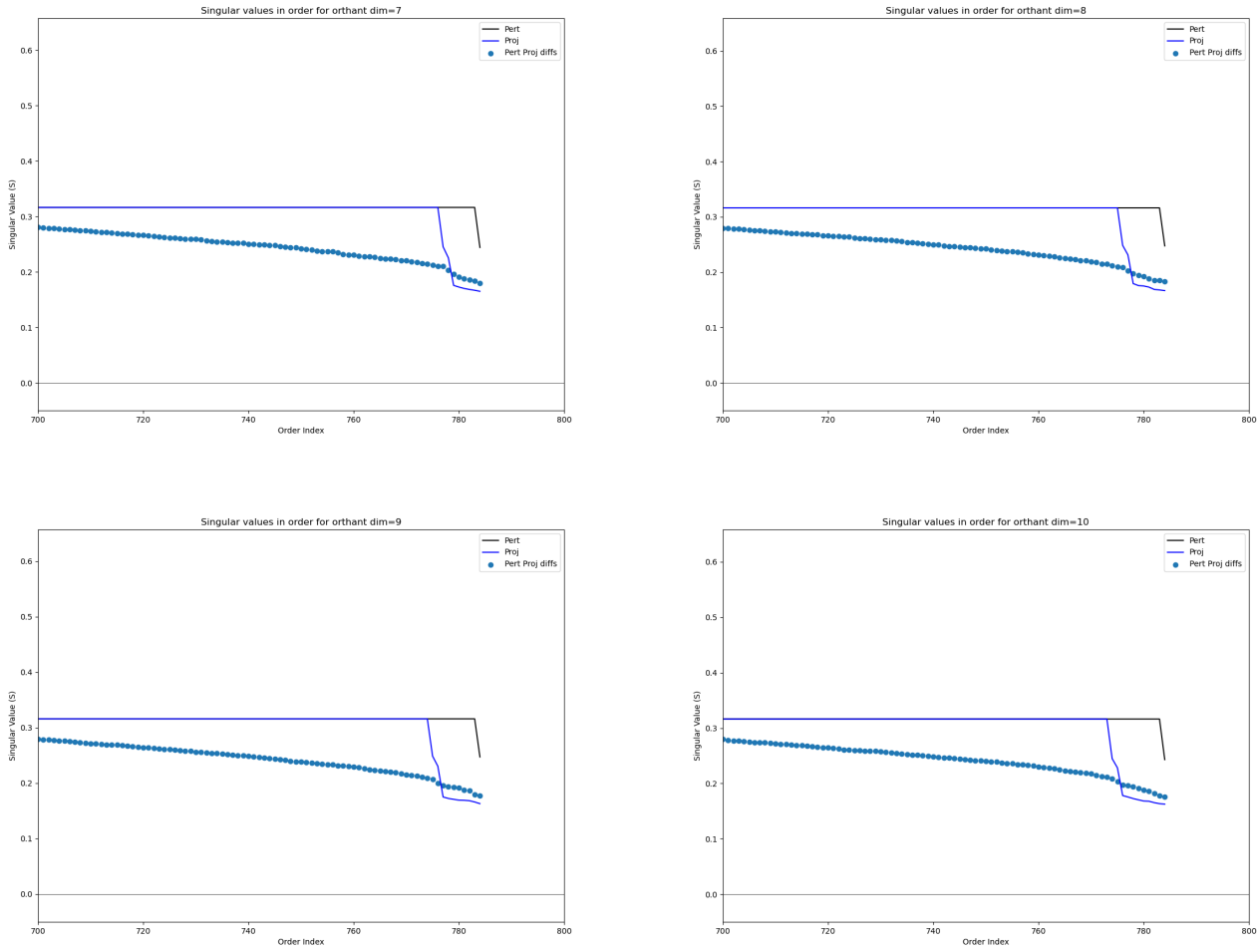

Singular values in order for orthant dim=2
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index


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

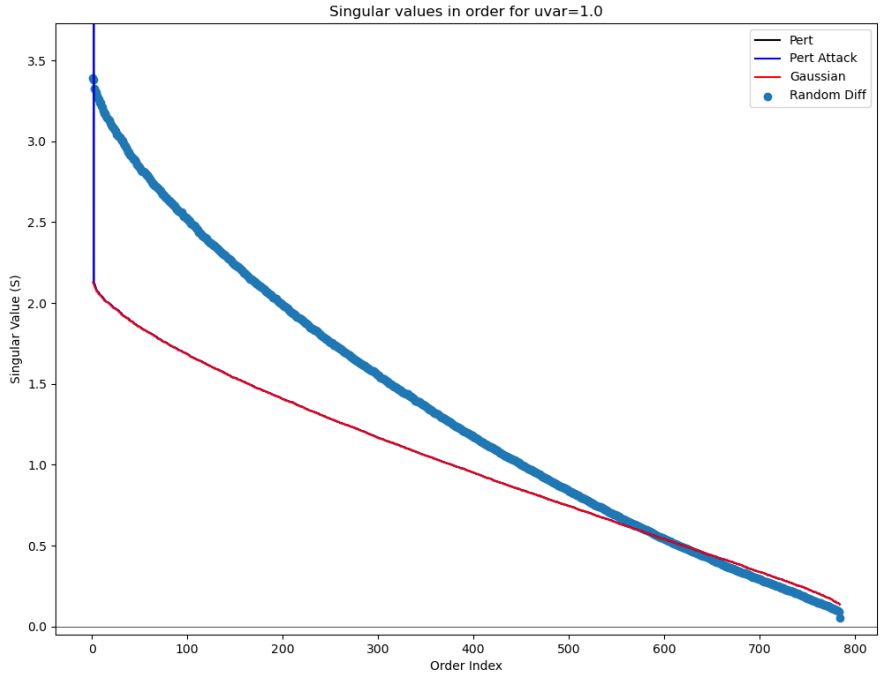


There is a question of how best to measure "normal" vectors for each projection. The most direct computation is to take a sample with small variance around each point on the decision boundary and solve least squares of each of these samples. We have previously observed that *most* of the decision boundary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

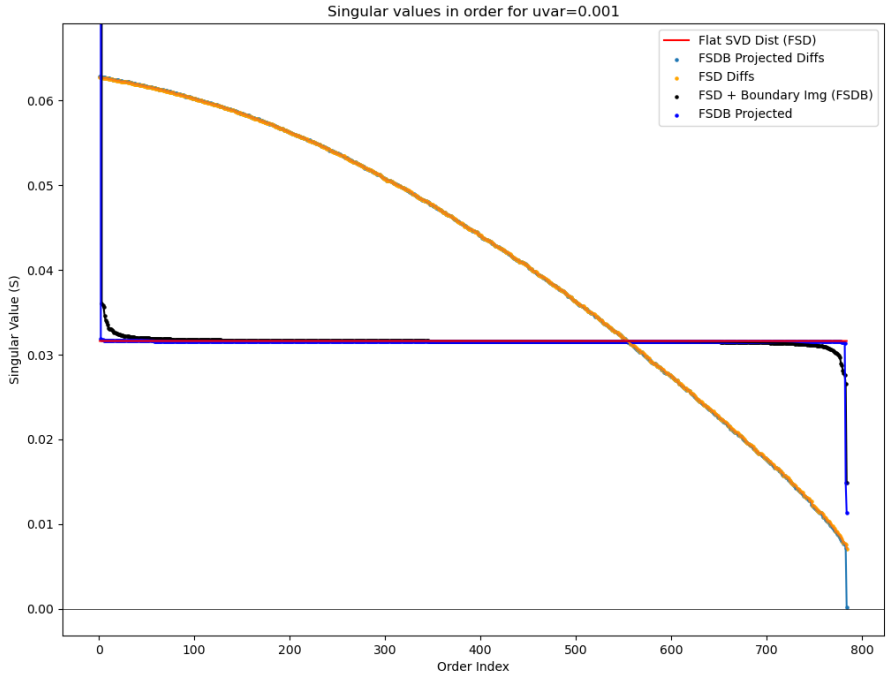
The following image shows singular values in order for a sample around a decision boundary image in MNIST generated by interpolating from a test image to an adversarial image generated from it. This plot is dominated by the natural distribution of singular values for a gaussian and also by the original 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.

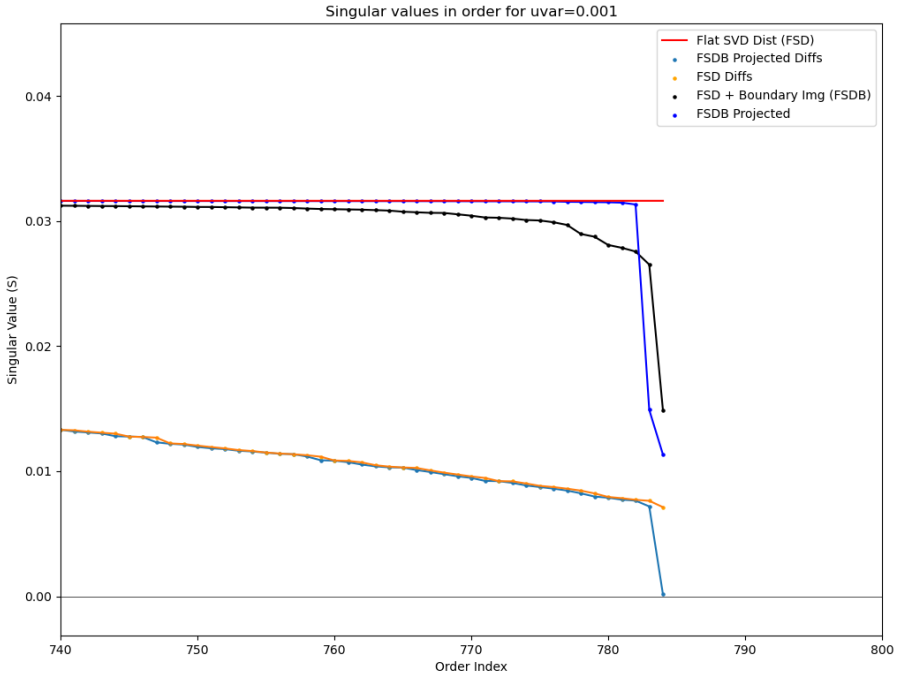


The following plots are generated with a new distribution to replace the gaussian. This "Flat SVD" distribution is generated by first taking a gaussian sample representable by a matrix X, then taking the SVD of this gaussian sample $U\Sigma V = X$, replacing Σ

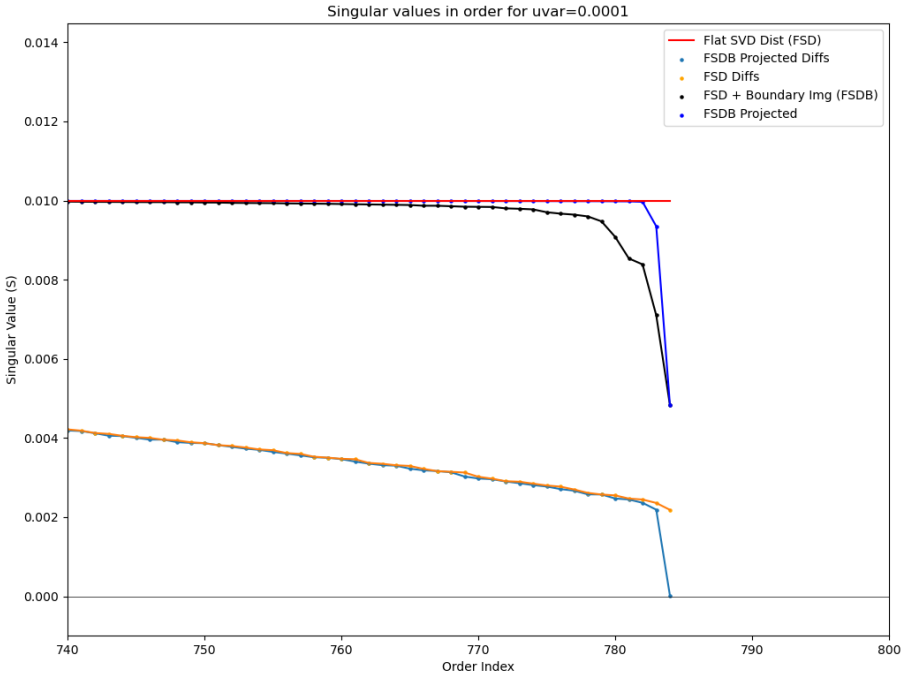
by $|\Sigma|_2 I$. This distribution has the property that its SVD is flat, while maintaining the Frobenius norm of X. This helps to highlight degeneracy in singular values.

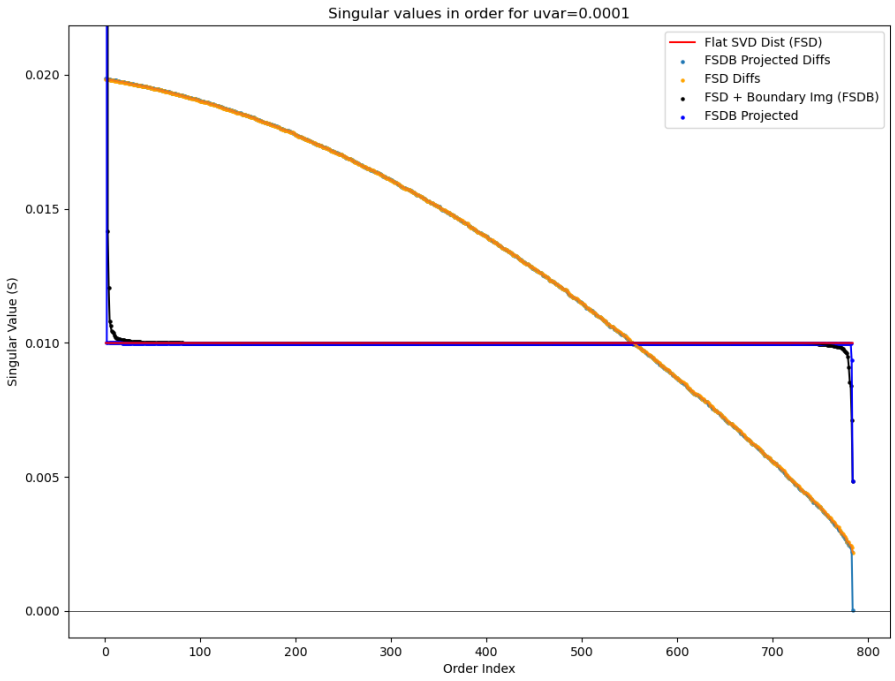
These first two plots are generated by sampling around a decision boundary image found by interpolating between two test images (rather than a test image and an adversarial example generated from it). We note that the SVD contains two degernate values. These seem to correpond with the two original images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





These last two images are generated with the flattened distribution

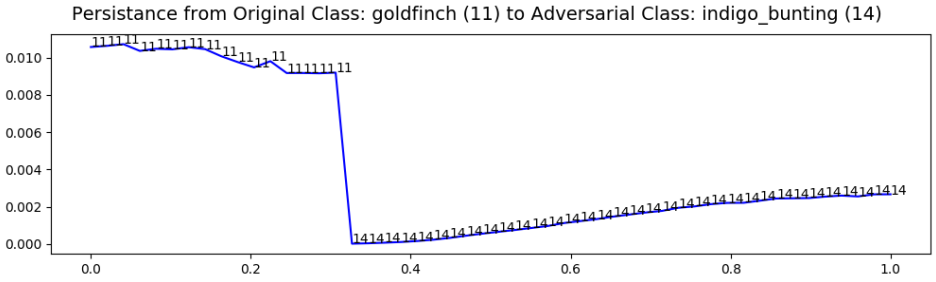

Singular values in order for uvar=0.0001
Flat SVD Dist (FSD)
FSDB Projected Diffs
FSD Diffs
FSD + Boundary Img (FSDB)
FSDB Projected
Singular Value (S)
Order Index
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
740
750
760
770
780
790
800



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.

2.7 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

2.7.1 in probability space

2.7.2 in image space

2.8 define orthants

2.9 skewness

2.10 sampling decision boundaries and analyzing dimensionality with PCA

2.11 neural network attack gradients versus decision boundaries

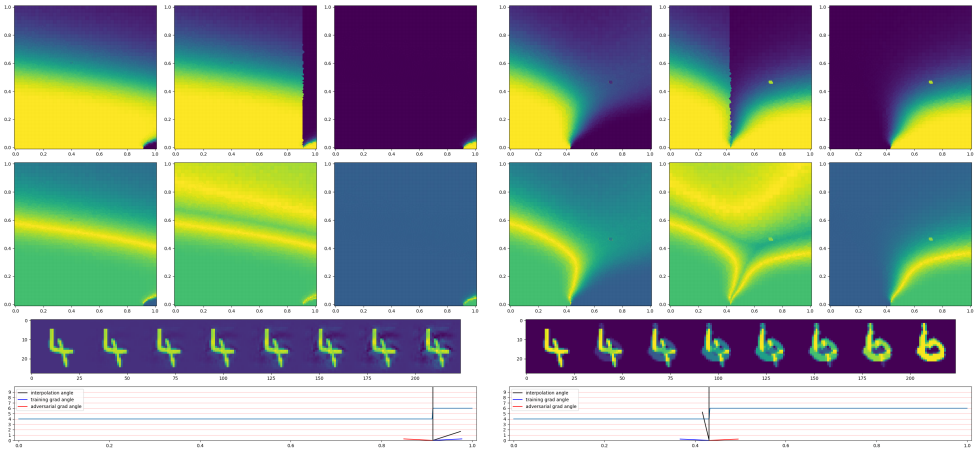
2.12 skewed orthant recreates persistence picture (?).

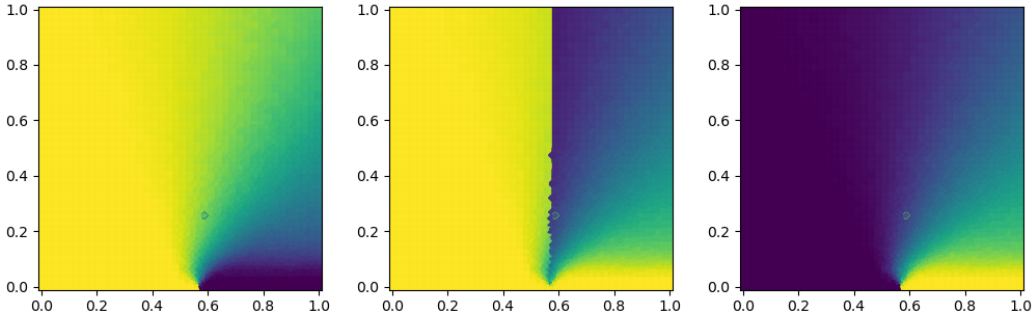
2.13 measuring skewness of ANNs

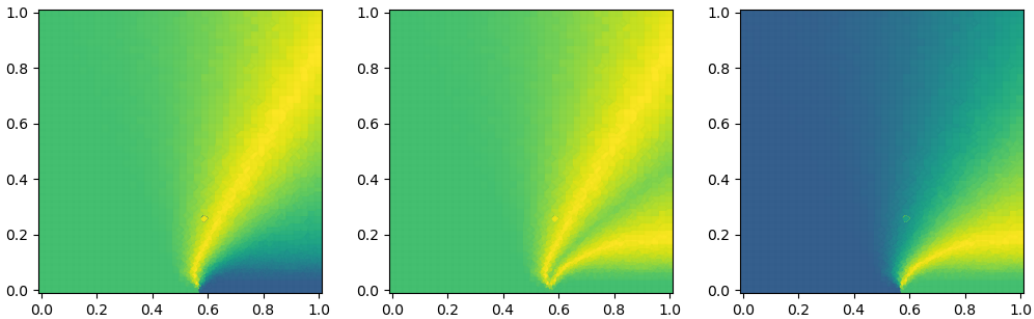
2.14 Relate skewness with dimpled manifold and features not bugs papers

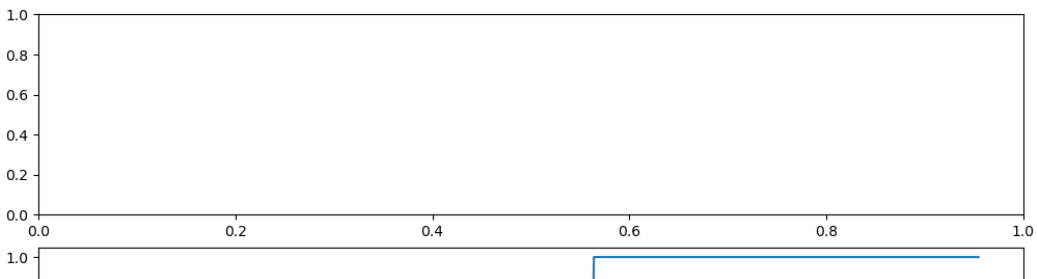
turn observations into are they a definition, a theorem, or a discussion

decision_boundary crossing




1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0 0.2 0.4 0.6 0.8 1.0

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 **??** 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 **??**

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 **??** 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 **??** 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 **??** and the networks from Table **??**. 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 these persistences for

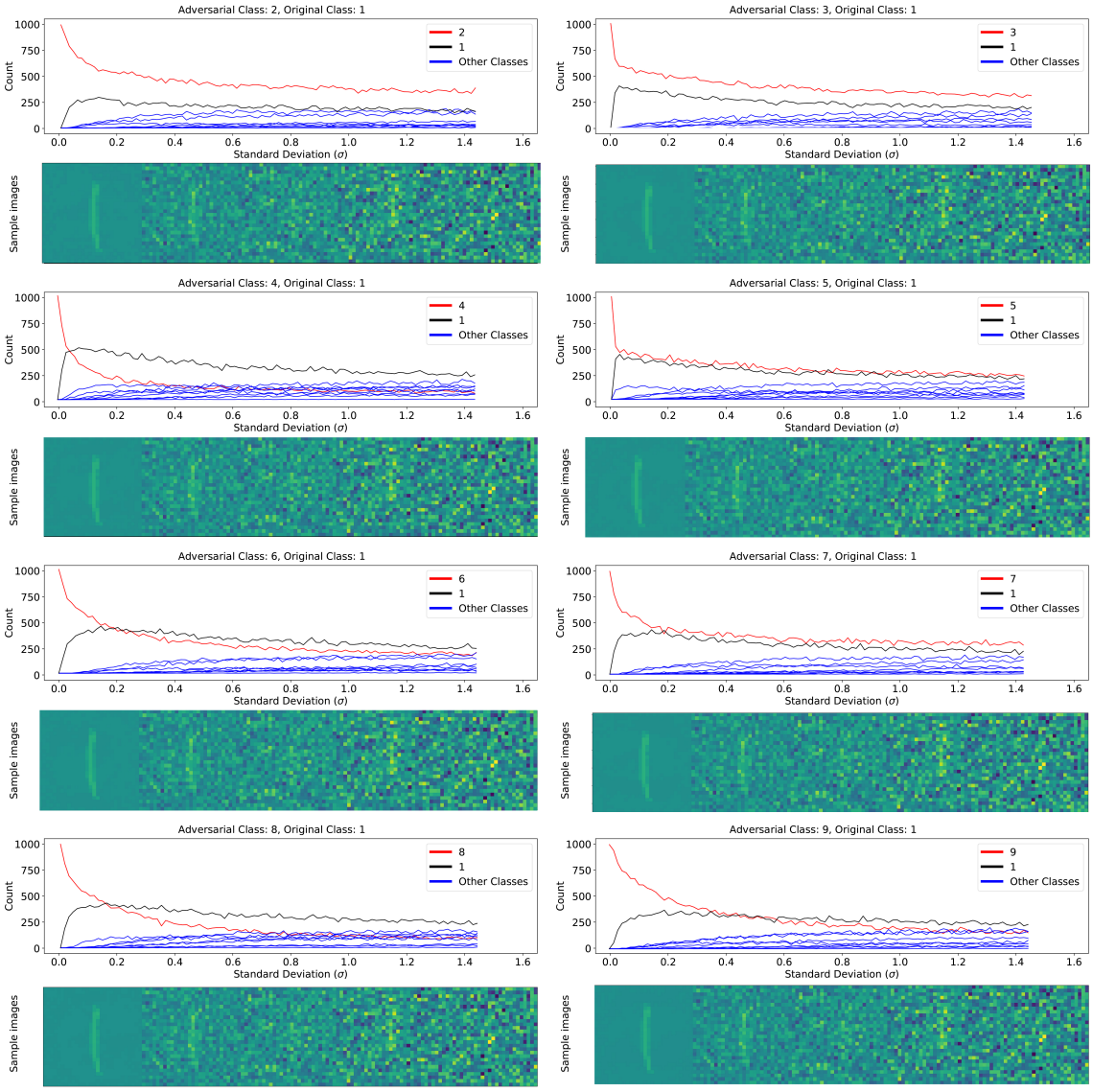


FIGURE B.1: Frequency of each class in Gaussian samples with increasing standard deviations around adversarial attacks of an image of a `1` targeted at classes `2` through `9` on a DNN classifier generated using IGSM. The adversarial class is shown as a red curve. The natural image class (`1`) is shown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

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 **??** and in Figure B.4 we see some of the convolutional neural networks from Table **??**.

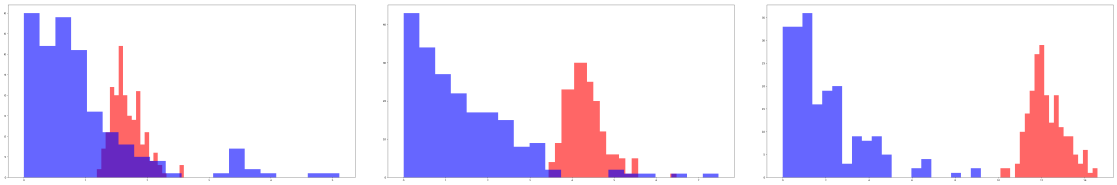


FIGURE B.2: Histograms of 0.7-persistence for FC10-4 (smallest regularization, left), FC10-2 (middle), and FC10-0 (most regularization, right) from Table **??**. Natural images are in blue, and adversarial images are in red. Note that these are plotted on different scales – higher regularization forces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

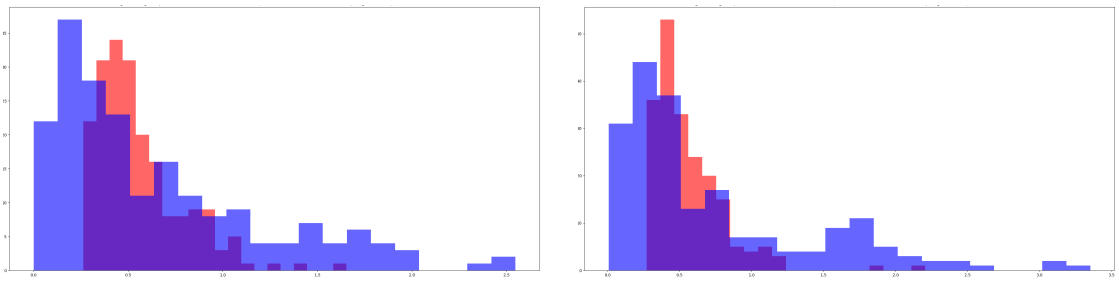


FIGURE B.3: Histograms of 0.7-persistence for FC100-100-10 (left) and FC200-200-10 (right) from Table **??**. Natural images are in blue, and adversarial images are in red.

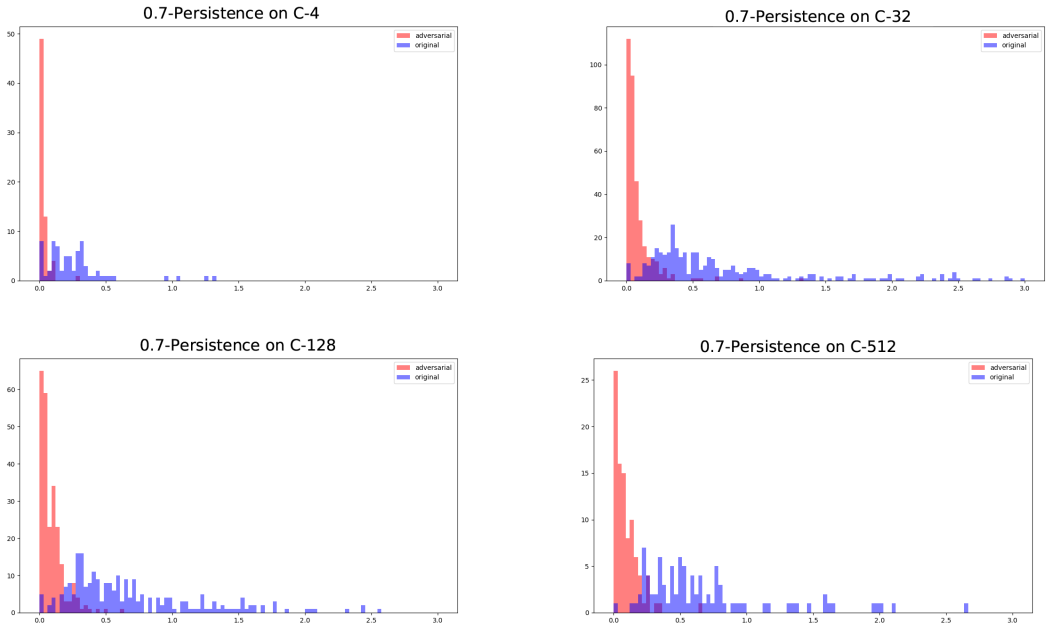


FIGURE B.4: Histograms of 0.7-persistence for C-4 (top left), C-32 (top right), C-128 (bottom left), and C-512 (bottom right) from Table **??**. Natural images are in blue and adversarial images are in red.

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 **??**, 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

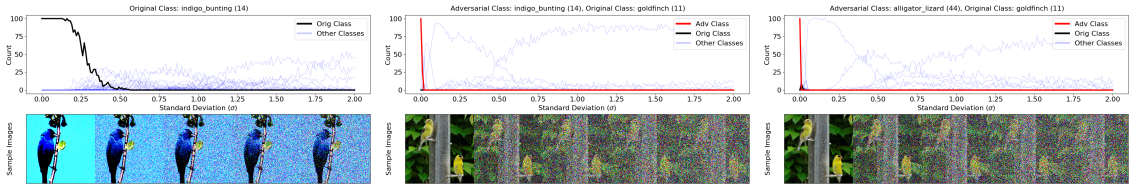


FIGURE B.5: Frequency of each class in Gaussian samples with increasing variance around an `indigo_bunting` image (left), an adversarial example of the image in class `goldfinch` from Figure **??** targeted at the `indigo_bunting` class on a alexnet network attacked with PGD (middle), and an adversarial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

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

Algorithm 1 Bracketing algorithm for computing γ-persistence

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

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

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

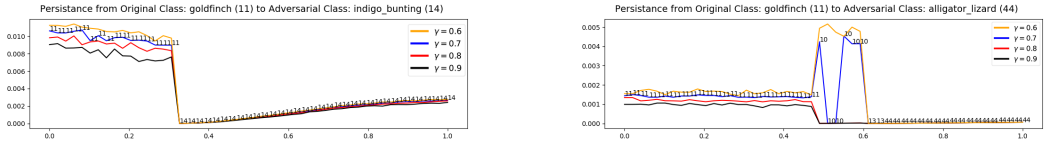


FIGURE B.6: The γ-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) (left) and to an adversarial image generated with PGL in the class `alligator_lizard` (44) (right) on a vgg16 classifier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

B.5 Concentration of measures

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

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

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

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

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

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

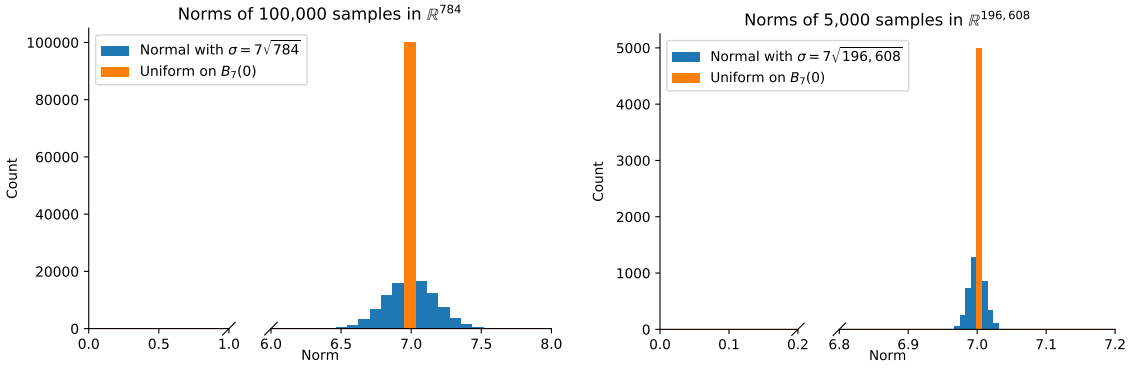


FIGURE B.7: Comparison of the length of samples drawn from $U(B_7(0))$ and $N(0, 7\sqrt{n})$ for $n = 784$, the dimension of MNIST, (left) and $n = 196,608$, the dimension of ImageNet, (right).

Appendix C The EPK is a Kernel

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

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

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

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

$$\tag{C.2}$$

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

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

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

$$\tag{C.5}$$

If $y_i^k = 0$ then we have

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

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

$$\tag{C.8}$$

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

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

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

This formulates a kernel machine where

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

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

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

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

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

□

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

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

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

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

In order for

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

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

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

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

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

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

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

$$\tag{C.24}$$

Otherwise, we get the usual

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

$$\tag{C.26}$$

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

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

C.0.2 Multi-Class Case

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

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

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

$$\tag{C.29}$$

Case 1 Scalar Loss

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

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

fashioned way on each row of that column:

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

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

Apply chain rule (C.32)

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

Let (C.34)

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

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

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

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

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

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

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

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

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

$$\tag{C.42}$$

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

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

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

$$\tag{C.45}$$

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

C.0.3 Schemes Other than Forward Euler (SGD)

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

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

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

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

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

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

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

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

C.0.4 Variance Estimation

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

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

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

$$\tag{C.51}$$

Where

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

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

Acknowledgements

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

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

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

Bibliography

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

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

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

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

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

Chen, Jiefeng et al. (2021a). "ATOM: Robustifying Out-of-Distribution Detection Using Outlier Mining". In: *Machine Learning and Knowledge Discovery in Databases. Research Track - European Conference, ECML PKDD 2021, Bilbao, Spain, September 13-17, 2021, Proceedings, Part III*. Ed. by Nuria Oliver et al. Vol. 12977. Lecture

Notes in Computer Science. Springer, pp. 430–445. DOI: 10.1007/978-3-030-86523-8_26. URL: https://doi.org/10.1007/978-3-030-86523-8_26.

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

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

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

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

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

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

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

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

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

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

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

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

Gong, Sixue, Vishnu Naresh Boddeti, and Anil K. Jain (2019). "On the Intrinsic Dimensionality of Image Representations". In: *IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019. Computer Vision Foundation / IEEE, pp. 3987–3996. DOI: 10.1109/CVPR.2019.00411. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Gong_On_the_Intrinsic_Dimensionality_of_Image_Representations_CVPR_2019_paper.html.

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

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

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

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

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

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

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

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

Jordan, P. and J. V. Neumann (1935). "On Inner Products in Linear, Metric Spaces". In: *Annals of Mathematics* 36.3, pp. 719–723. ISSN: 0003486X. URL: http://www.jstor.org/stable/1968653 (visited on 07/21/2023).

Kaufman, Ilya and Omri Azencot (July 3, 2023). "Data Representations' Study of Latent Image Manifolds". In: *Proceedings of the 40th International Conference on*

Machine Learning. International Conference on Machine Learning. ISSN: 2640-3498. PMLR, pp. 15928–15945. URL: https://proceedings.mlr.press/v202/kaufman23a.html (visited on 08/02/2023).

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.

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

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

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

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

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

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

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

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

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

Shamir, Adi, Odelia Melamed, and Oriel BenShmuel (2021). "The dimpled manifold model of adversarial examples in machine learning". In: *arXiv preprint arXiv:2106.10151*.

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

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

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

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

Srinivas, Suraj, Sebastian Bordt, and Hima Lakkaraju (2023). "Which Models have Perceptually-Aligned Gradients? An Explanation via Off-Manifold Robustness". In:

CoRR abs/2305.19101. DOI: 10.48550/arXiv.2305.19101. arXiv: 2305.19101. URL: https://doi.org/10.48550/arXiv.2305.19101.

Sun, Yiyou, Chuan Guo, and Yixuan Li (2021). "ReAct: Out-of-distribution Detection With Rectified Activations". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al., pp. 144–157. URL: https://proceedings.neurips.cc/paper/2021/hash/01894d6f048493d2cacde3c579c315a3-Abstract.html.

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

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

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

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

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

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

Wegner, Sven-Ake (2021). "Lecture notes on high-dimensional spaces". In: *arXiv preprint arXiv:2101.05841*.

Xu, Mingyu et al. (2023). *VRA: Variational Rectified Activation for Out-of-distribution Detection*. arXiv: 2302.11716 [cs.LG].

Yang, Huanrui et al. (2020). "Learning Low-rank Deep Neural Networks via Singular Vector Orthogonality Regularization and Singular Value Sparsification". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*. Computer Vision Foundation / IEEE, pp. 2899–2908. DOI: 10.1109/CVPRW50498.2020.00347. URL: https://openaccess.thecvf.com/content_CVPRW_2020/html/w40/Yang_Learning_Low-Rank_Deep_Neural_Networks_via_Singular_Vector_Orthogonality_Regularization_CVPRW_2020_paper.html.

Yang, Jingkang et al. (2021). "Generalized Out-of-Distribution Detection: A Survey". In: *CoRR* abs/2110.11334. arXiv: 2110.11334. URL: https://arxiv.org/abs/2110.11334.

Yousefzadeh, Roozbeh (2021). "Deep learning generalization and the convex hull of training sets". In: *arXiv preprint arXiv:2101.09849*.

Zheng, Yijia et al. (2022). "Learning Manifold Dimensions with Conditional Variational Autoencoders". en. In: *Advances in Neural Information Processing Systems* 35, 34709–34721.