

A GEOMETRIC FRAMEWORK FOR ADVERSARIAL VULNERABILITY IN  
MACHINE LEARNING

by

Brian Bell

---

Copyright © Brian Bell 2023

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 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**

Date: Type defense

(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

|   |           |
|---|-----------|
| <b>Acknowledgements</b>   | <b>4</b>  |
| <b>Quotations</b>   | <b>6</b>  |
| <b>List of Figures</b>  | <b>9</b>  |
| <b>List of Tables</b>   | <b>14</b> |
| <b>List of Abbreviations</b>  | <b>15</b> |
| <b>Physical Constants</b>   | <b>16</b> |
| <b>List of Symbols</b>  | <b>17</b> |
| <b>Abstract</b>   | <b>18</b> |
| 0.1 Introduction . . . . .  | 20        |
| 0.2 Related Work . . . . .  | 22        |
| 0.3 Theoretical Justification : Generalized Exact Path Kernel . . . . . | 24        |
| 0.4 OOD is enabled by Parameter Gradients . . . . .                     | 27        |
| 0.4.1 Expressing Prior OOD Methods with the gEPK . . . . .              | 28        |
| 0.4.2 gEPK for OOD . . . . .  | 32        |
| 0.5 Signal Manifold Dimension Estimated with Training Input Gradients   | 33        |
| 0.6 Conclusion . . . . .  | 37        |

|   |           |
|---|-----------|
| <b>A Attacks</b>  | <b>41</b> |
| A.1 L-BFGS . . . . .  | 41        |
| <b>B Persistence Tools</b>  | <b>43</b> |
| B.1 Bracketing Algorithm . . . . .  | 43        |
| B.2 Bracketing Algorithm . . . . .  | 43        |
| B.3 Convolutional neural networks used . . . . .                                | 43        |
| B.4 Additional Figures . . . . .  | 45        |
| B.4.1 Additional figures from MNIST . . . . .                                   | 45        |
| B.4.2 Additional figures for ImageNet . . . . .                                 | 48        |
| B.5 Concentration of measures . . . . .   | 51        |
| <b>C The EPK is a Kernel</b>  | <b>53</b> |
| C.0.1 When is an Ensemble of Kernel Machines itself a Kernel Machine? . . . . . | 53        |
| C.0.2 Multi-Class Case . . . . .  | 57        |
| Case 1 Scalar Loss . . . . .  | 57        |
| C.0.3 Schemes Other than Forward Euler (SGD) . . . . .                          | 60        |
| C.0.4 Variance Estimation . . . . .   | 61        |
| <b>Bibliography</b>   | <b>63</b> |

## List of Figures

- |   |   |              |
|---|---|--------------|
| 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. | . . . . . 21 |
| 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).   | . . . . . 27 |



|  |    |
|--|----|
| 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). . . . . | 40 |
| 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. . . . .   | 46 |
| 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. . . . .  | 47 |

|  |    |
|--|----|
| 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. . . . .  | 47 |
| 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. . . . .  | 48 |
| B.5 Frequency of each class in Gaussian samples with increasing variance around an <code>indigo_bunting</code> image (left), an adversarial example of the image in class <code>goldfinch</code> from Figure ?? targeted at the <code>indigo_bunting</code> class on a alexnet network attacked with PGD (middle), and an adversarial example of the <code>goldfinch</code> image targeted at the <code>alligator_lizard</code> class on a vgg16 network attacked with PGD (right). Bottoms show example sample images at different standard deviations. . . . .   | 49 |
| B.6 The $\gamma$ -persistence of images along the straight line path from an image in class <code>goldfinch</code> (11) to an adversarial image generated with BIM in the class <code>indigo_bunting</code> (14) (left) and to an adversarial image generated with PGL in the class <code>alligator_lizard</code> (44) (right) on a vgg16 classifier with different values of $\gamma$ . 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 $\gamma$ -persistence and the horizontal axis is progress towards the adversarial image. . . . . | 51 |



## List of Tables

|   |    |
|---|----|
| B.1 Structure of the CNNs C-Ch used in Table ?? . . . . . | 45 |
|---|----|

## List of Abbreviations

**LAH** List Abbreviations Here  
**WSF** What (it) Stands For

## Physical Constants

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

## List of Symbols

|          |                   |                         |
|----------|-------------------|-------------------------|
| $a$      | distance          | m                       |
| $P$      | power             | W ( $\text{J s}^{-1}$ ) |
| $\omega$ | 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...

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.

## 0.1 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*

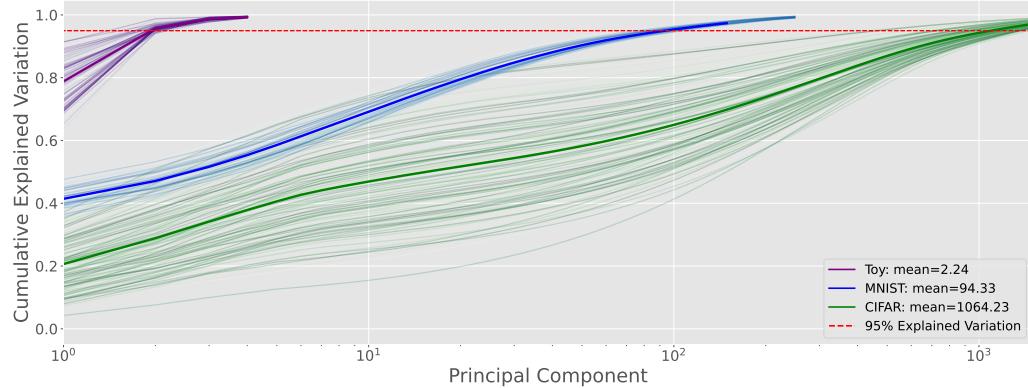


FIGURE 1: The gEPK naturally provides a measure of input dimension. This plot shows the CDF of the explained variation of training point sensitivities  $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$ . Different datasets are color coded to show differences in signal dimension. 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 0.3 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.

## 0.2 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 0.4 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.

### 0.3 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 0.3.1** (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  $\theta_0$ . Then for an arbitrary test point  $x$ , the trained*

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

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

$$\varphi_{s,t}(x) \equiv \nabla_{\theta} f(x; \theta_s(t)), \quad (2)$$

$$\theta_s(t) \equiv \theta_s(0) + t(\theta_{s+1}(0) - \theta_s(0)), \text{ and} \quad (3)$$

$$\hat{y}_{\theta_s(0)} \equiv f(x; \theta_s(0)). \quad (4)$$

*Proof.* Guided by the proof for Theorem 6 from Bell et al. (2023), let  $\theta$  and  $f(\cdot; \theta)$  satisfy the conditions of Theorem 0.3.1, 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  $\theta_s$  to  $\theta_{s+1}$ . Since we are training using gradient descent, we can write  $\theta_{s+1} \equiv \theta_s + \frac{d\theta_s(t)}{dt}$ . We derive a linear interpolate connecting these states:

$$\frac{d\theta_s(t)}{dt} = (\theta_{s+1} - \theta_s) \quad (5)$$

$$\int \frac{d\theta_s(t)}{dt} dt = \int (\theta_{s+1} - \theta_s) dt \quad (6)$$

$$\theta_s(t) = \theta_s + t(\theta_{s+1} - \theta_s) \quad (7)$$

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

$$\frac{d\theta_s(t)}{dt} = -\varepsilon \nabla_w L(f(X_T; \theta_s(0)), y_i) = -\sum_{i=1}^N \varepsilon \sum_{j=1}^M \frac{\partial L(f(x_i; \theta_s(0)), y_i)}{\partial \theta^j} \quad (8)$$

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

$$\frac{d\hat{y}}{dt} = \frac{df(x; \theta_s(t))}{dt} = \sum_{j=1}^M \frac{df}{\partial \theta^j} \frac{\partial \theta^j}{dt} = \sum_{j=1}^M \frac{df(x; \theta_s(t))}{\partial \theta^j} \left( -\varepsilon \frac{\partial L(f(X_T, \theta_s(0)), Y_T)}{\partial \theta^j} \right) \quad (9)$$

$$= \sum_{j=1}^M \frac{df(x; \theta_s(t))}{\partial \theta^j} \left( - \sum_{i=1}^N \varepsilon \frac{dL(f(x_i; \theta_s(0)), y_i)}{df(x_i; \theta_s(0))} \frac{\partial f(x_i; \theta_s(0))}{\partial \theta^j} \right) \quad (10)$$

$$= - \sum_{i=1}^N \varepsilon \nabla_w f(x; \theta_s(t)) \frac{dL(f(x_i; \theta_s(0)), y_i)}{df(x_i; \theta_s(0))} \cdot \nabla_w f(x_i; \theta_s(0)) \quad (11)$$

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

$$y_{s+1} - y_s = \int_0^1 - \sum_{i=1}^N \varepsilon \frac{dL(f(x_i; \theta_s(0)), y_i)}{df(x_i; \theta_s(0))} \nabla_w f(x; \theta_s(t)) \cdot \nabla_w f(x_i; \theta_s(0)) dt \quad (12)$$

$$= - \sum_{i=1}^N \varepsilon \left( \int_0^1 \nabla_w f(x; \theta_s(t)) dt \right) \frac{dL(f(x_i; \theta_s(0)), y_i)}{df(x_i; \theta_s(0))} \cdot \nabla_w f(x_i; \theta_s(0)) \quad (13)$$

For all  $N$  training steps, we have

$$y_N = f(x; \theta_0) + \sum_{s=1}^N y_{s+1} - y_s \quad (14)$$

$$= f(x; \theta_0) - \sum_{s=1}^N \sum_{i=1}^N \varepsilon \left( \int_0^1 \nabla_w f(x; \theta_s(t)) dt \right) \frac{dL(f(x_i; \theta_s(0)), y_i)}{df(x_i; \theta_s(0))} \cdot \nabla_w f(x_i; \theta_s(0)) \quad (15)$$

□

**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.

## 0.4 OOD is enabled by Parameter Gradients

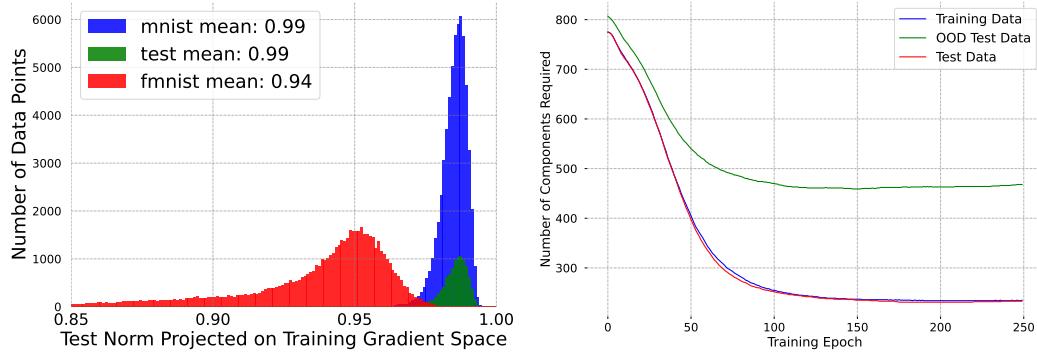


FIGURE 2: OOD detection using difference in training vs. test gradients. As the purpose of this paper is not to develop state of the art OOD detection methods, a comparison with recent benchmarks is not provided. Instead, a proof of concept that the gEPK can perform OOD detection is given. Left histogram 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).

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

$$\left\{ \frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i} \varphi_{s,0}(x_i); i \in \{1, \dots, N\}, s \in \{1, \dots, S\} \right\} \quad (16)$$

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. 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.

#### 0.4.1 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:

$$\frac{1}{C} \sum_i^C \frac{\partial L_{CE}(f(x; \theta), i)}{\partial \hat{y}} \nabla_{\theta} f(x; \theta) \quad (17)$$

This looks like the left side of the inner product from the gEPK, however the scaling factor,  $\frac{\partial L_{CE}(f(x; \theta), i)}{\partial \hat{y}}$ , 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 0.1 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

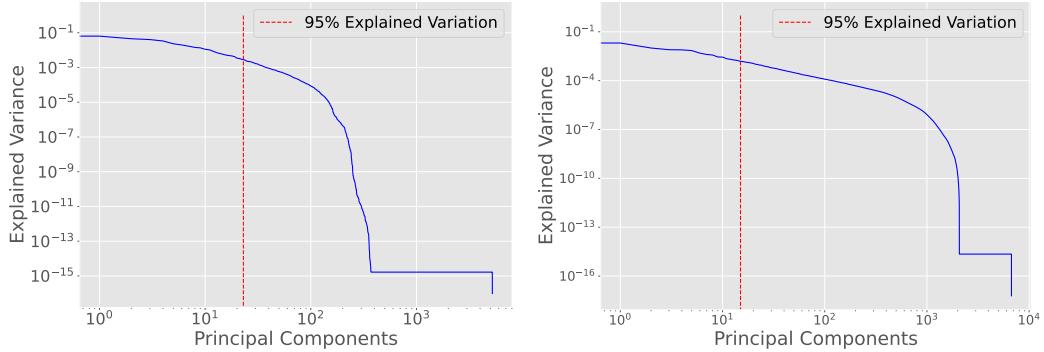


FIGURE 3: Explained Variance Ratio of parameter gradients. Left: MNIST, Right: CIFAR. 95% of variation can be explained with a relatively low number of components in both cases.

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  $p^{\text{th}}$  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:

$$\nabla_{\theta} L(x, y) = (\theta x - y)x^T = \Omega x^T \quad (18)$$

Treating  $\Omega$  as an error vector, they prove that all variation of the output must be 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  $R_{ID}^L = [x_1, x_2, \dots, x_n]$  over which they compute the SVD,  $U_{ID}^L \Sigma_{ID}^L (V_{ID}^L)^T = R_{ID}^L$ . This representation is then truncated to  $k$  principal components according to a threshold  $\epsilon_{th}$  such that

$$\left\| U_{ID}^L \Sigma_{ID,k}^L (V_{ID}^L)^T \right\|_F^2 \geq \epsilon_{th} \|R_{ID}^L\|_F^2. \quad (19)$$

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

$$O(x) = (\nabla_{\theta_L} \mathcal{L}(f(x, \theta_L), y)) S^L (S^L)^T \quad (20)$$

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  $\theta_L$  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.

### 0.4.2 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:

$$\text{span} \left( \left\{ \frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i} \varphi_{s,0}(x_i) : x_i \in X_T, 0 \leq s \leq S \right\} \right). \quad (21)$$

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. 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.

## 0.5 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:

$$f(x, \theta_{\text{trained}}) = f(x, \theta_0(0)) + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) dt \quad (22)$$

$$\varphi_{s,t}(x) = \nabla_\theta f(x, \theta_s(t)) \quad (23)$$

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  $x_i$ :

$$\frac{df(x, \theta_{\text{trained}})}{dx_j} = \frac{df(x, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d \left( \varphi_{s,t}(x) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) \right)}{dx_j} dt \quad (24)$$

$$= 0 + \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) \frac{d \left( \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) \right)}{dx_j} dt \quad (25)$$

$$= \sum_i \sum_s \int_0^1 \varphi_{s,t}(x) dt \left( \frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \quad (26)$$

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

$$G_j = \sum_s \int_0^1 \varphi_{s,t}(x) dt \left( \frac{d^2 L(x_i, y_i)}{df(x_i, \theta_s(0)) dx_j} \varphi_{s,0}(x_i) + \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \frac{d\varphi_{s,0}(x_i)}{dx_j} \right) \quad (27)$$

While written in this form, it appears we must keep second-order derivatives, however we note that the inner product with  $\varphi_{s,t}(x)$  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  $\text{span}(G)$  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  $60000 \times 784$  (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 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 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.

$$\frac{df(x_j, \theta_{\text{trained}})}{dx_j} = \frac{df(x_j, \theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d \left( \varphi_{s,t}(x_j) \frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) \right)}{dx_j} dt \quad (28)$$

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  $H_{\theta_0}$ . By comparing the rank and span of  $H_{\theta_{\text{trained}}}$  and  $H_{\theta_0}$  we can understand to what extent the model's spatial representation of the data is due to the initial parameter selection and how much is due to the training path. Also,  $H_{\theta_{\text{trained}}}$  provides sample of gradients across all training data, which in some sense must be spanned by the model's implicit subspace basis. Despite missing the granular subspace information, the rank of this gradient matrix and its explained variation computed using SVD should be related to the model's implicit subspace rank. It should be noted that while there is a direct relationship between a models variations in input space and weight space, Figure 5 shows that this mapping changes greatly from the beginning to end of training and that this spectrum starts out wide (high dimensional) for  $\theta_0$  and much more focused (low dimensional) for  $\theta_T$ .

One interesting property of using input gradients for training data decomposed according to Eq. 27 is the ability to compare input gradients across models with different initial parameters and even different architectures. Figure 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.

## 0.6 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.

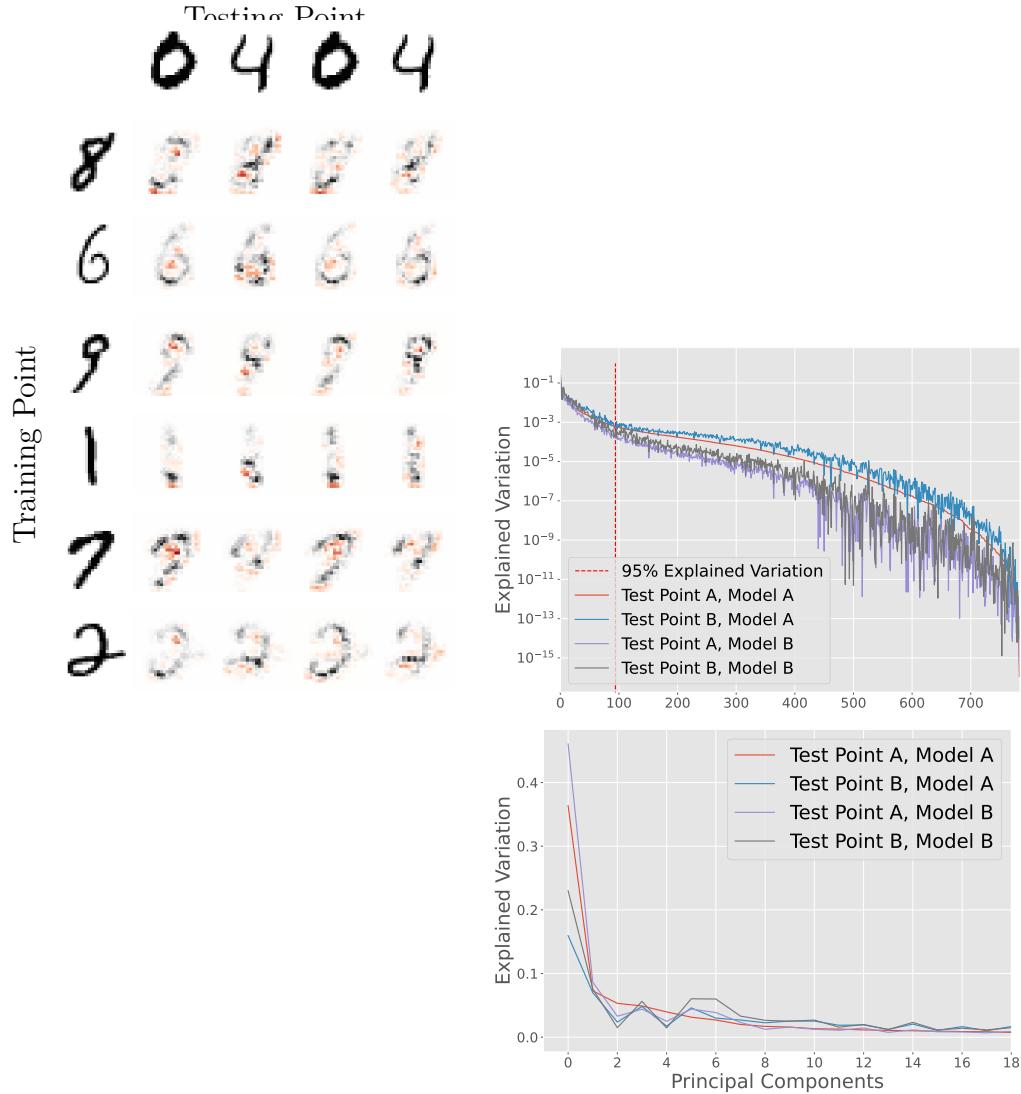


FIGURE 4: Left: Visualization of training point input gradients on test points compared between two models. Positive contribution (black) and negative contribution (red) of each training datum to the prediction for each test point. Elements in the grid are  $\nabla_{x_{\text{train}}} f(x_{\text{test}}, \theta_{\text{trained}})$ . Right: By taking these individual gradient contributions for a test point and computing the SVD across the training, the significant modes of variation in the input space can be measured (sigma squared). Top is log scale of the full spectrum, bottom shows the first 10 components. Note that this decomposition selects similar, but not identical, modes of variation across test points and even across different models. Components in SVD plots are sorted using Test Point A on Model A.

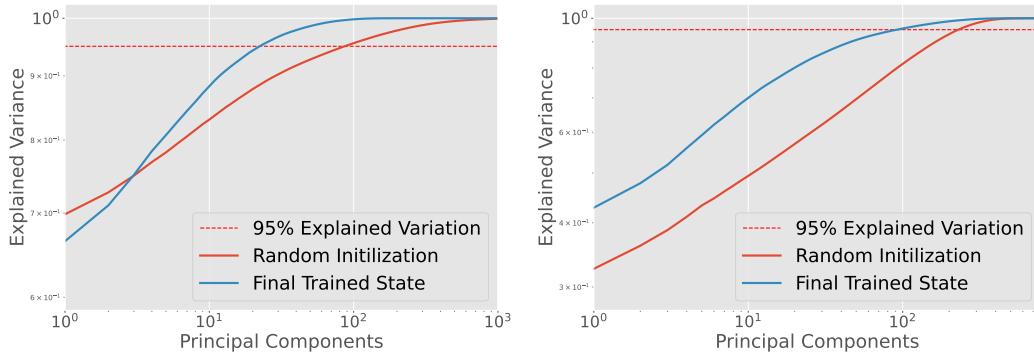


FIGURE 5: Differences between observing gradients in input space vs. weight space. Left: CDF of explained variation parameter space. Right: CDF of explained variation input space. Red solid line indicates a model at random initialization while the blue solid line represents the fully trained state. From 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).

## 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  $k = (\text{the number of iterations so far})$  **do**

$$d_k = -H_k g_k,$$

$x_{k+1} = x_k + \alpha_k d_k$ , Where  $\alpha_k$  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) | (Ch, 24, 24) |
| 2     | Conv     | Ch                | (5, 5) | (1, 1) | (Ch, 20, 20) |
| 3     | Conv     | Ch                | (5, 5) | (1, 1) | (Ch, 16, 16) |
| 4     | Conv     | Ch                | (5, 5) | (1, 1) | (Ch, 12, 12) |
| 5     | Max Pool | Ch                | (2, 2) | (2, 2) | (Ch, 6, 6)   |
| 7     | FC       | (Ch · 6 · 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

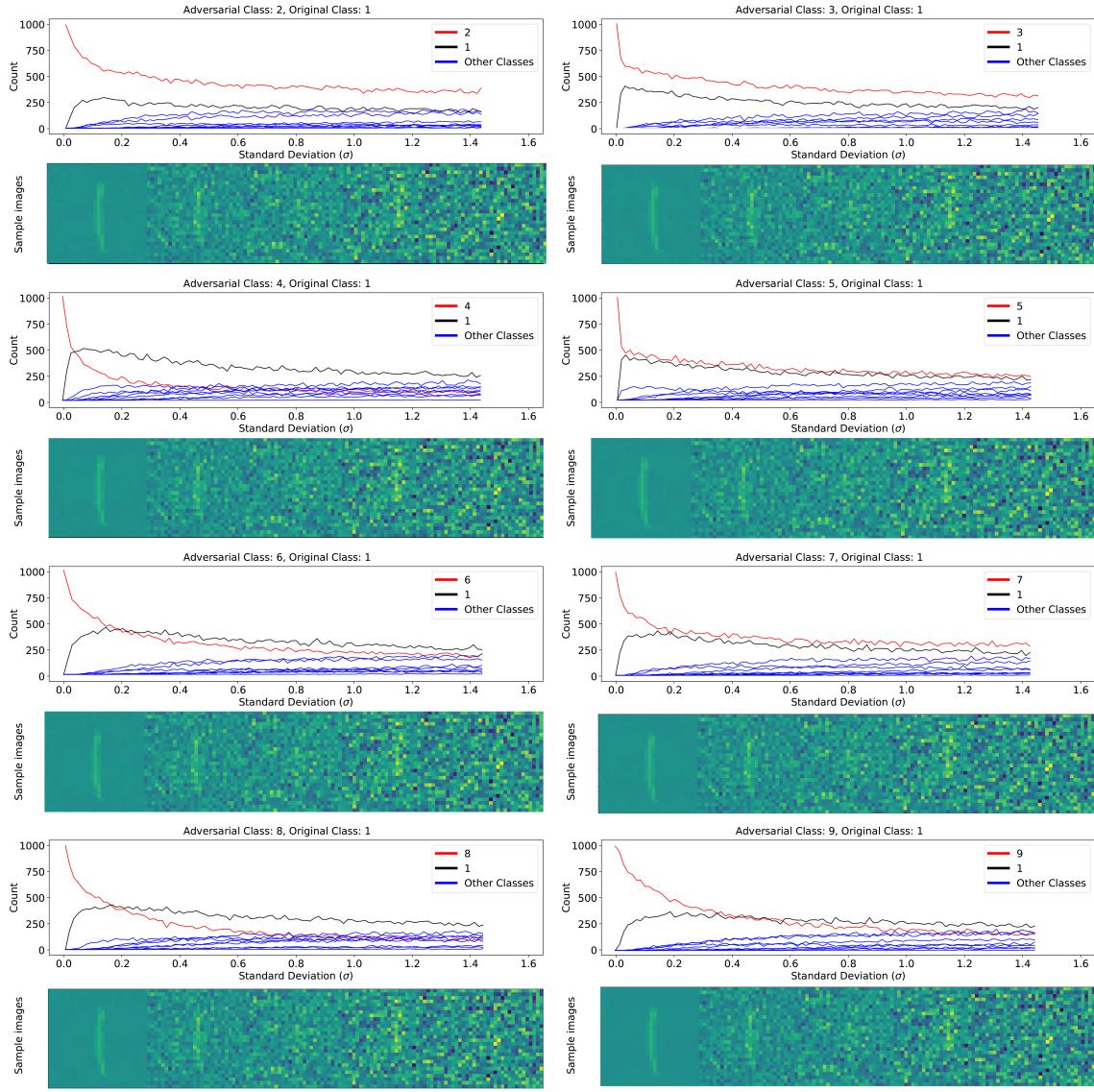


FIGURE B.1: Frequency of each class in Gaussian samples with increasing standard deviations around adversarial attacks of an image of a 1 targeted at classes 2 through 9 on a DNN classifier generated using IGSM. The adversarial class is shown as a red curve. The natural image class (1) is shown in black.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

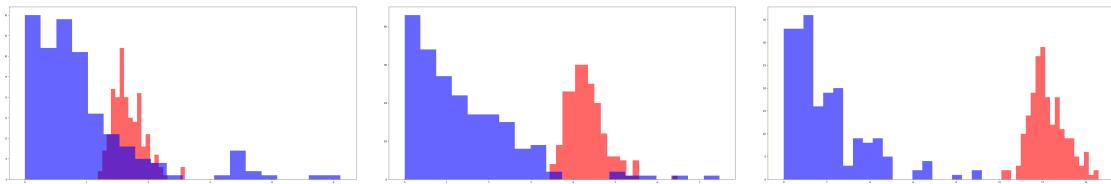


FIGURE B.2: Histograms of 0.7-persistence for FC10-4 (smallest regularization, left), FC10-2 (middle), and FC10-0 (most regularization, right) from Table ???. Natural images are in blue, and adversarial images are in red. Note that these are plotted on different scales – higher regularization forces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

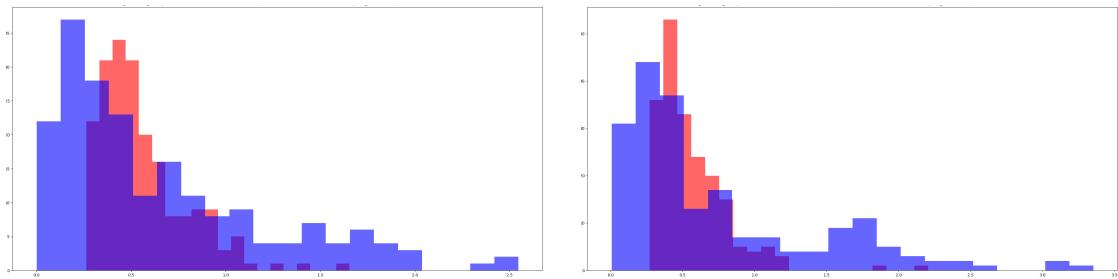


FIGURE B.3: Histograms of 0.7-persistence for FC100-100-10 (left) and FC200-200-10 (right) from Table ???. Natural images are in blue, and adversarial images are in red.

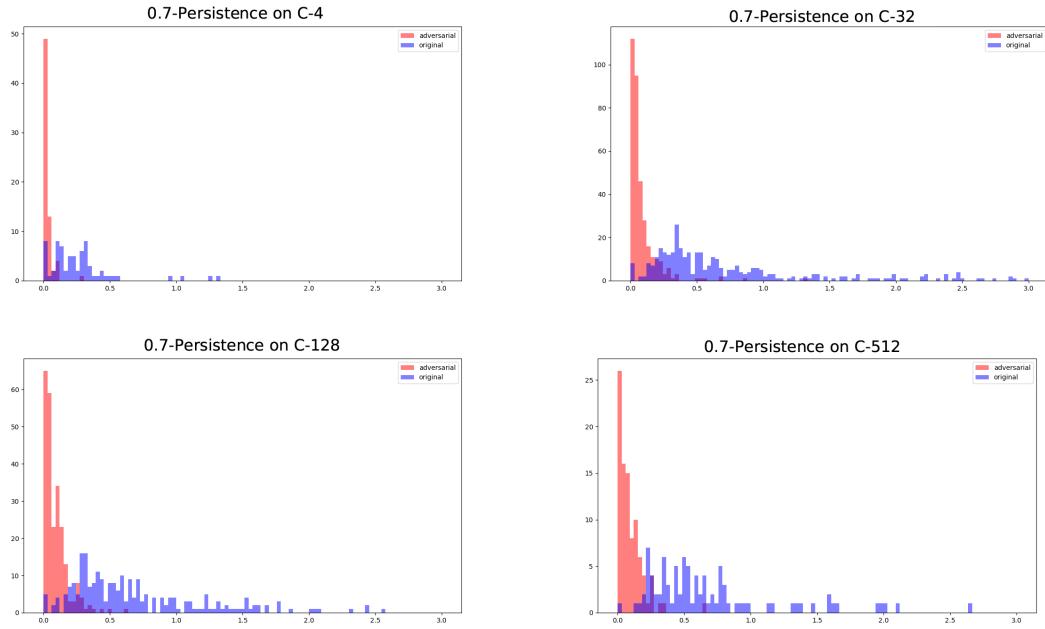


FIGURE B.4: Histograms of 0.7-persistence for C-4 (top left), C-32 (top right), C-128 (bottom left), and C-512 (bottom right) from Table ???. Natural images are in blue and adversarial images are in red.

#### 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  $\gamma$ -persistence along a straight line from a natural

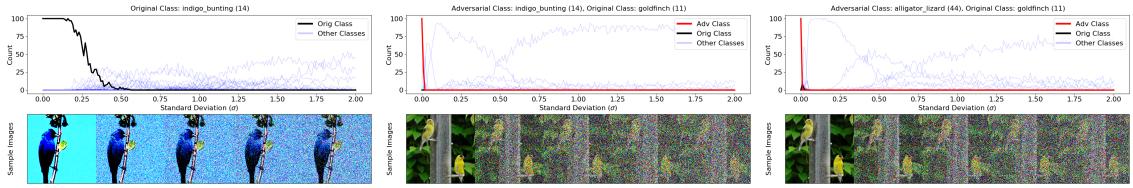


FIGURE B.5: Frequency of each class in Gaussian samples with increasing variance around an `indigo_bunting` image (left), an adversarial example of the image in class `goldfinch` from Figure ?? targeted at the `indigo_bunting` class on a alexnet network attacked with PGD (middle), and an adversarial 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  $\gamma$ . The  $\gamma$ -persistence in each case seems to change primarily when crossing the decision boundary. Interestingly, while the choice of  $\gamma$  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  $\gamma$  value, and that persistence does indeed depend on the landscape of the decision boundary described by the classifier.

---

**Algorithm 1** Bracketing algorithm for computing  $\gamma$ -persistence

---

```

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

function RANGEFINDER(image,  $\mathcal{C}$ , numSamples,  $\gamma$ )
     $\sigma_{\min} = .5, \sigma_{\max} = 1.5$ 
     $\gamma_1 = \text{COMPUTE\_PERSISTENCE}(\sigma_{\min}, \text{image}, \text{numSamples}, \mathcal{C})$ 
     $\gamma_2 = \text{COMPUTE\_PERSISTENCE}(\sigma_{\max}, \text{image}, \text{numSamples}, \mathcal{C})$ 
    while  $\gamma_1 < \gamma$  or  $\gamma_2 > \gamma$  do
        if  $\gamma_1 < \gamma$  then
             $\sigma_{\min} = .5\sigma_{\min}$ 
             $\gamma_1 = \text{COMPUTE\_PERSISTENCE}(\sigma_{\min}, \text{image}, \text{numSamples}, \mathcal{C})$ 
        end if
        if  $\gamma_2 > \gamma$  then
             $\sigma_{\max} = 2\sigma_{\max}$ 
             $\gamma_2 = \text{COMPUTE\_PERSISTENCE}(\sigma_{\max}, \text{image}, \text{numSamples}, \mathcal{C})$ 
        end if
    end while
    return  $[\sigma_{\min}, \sigma_{\max}]$ 
end function

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

```

---

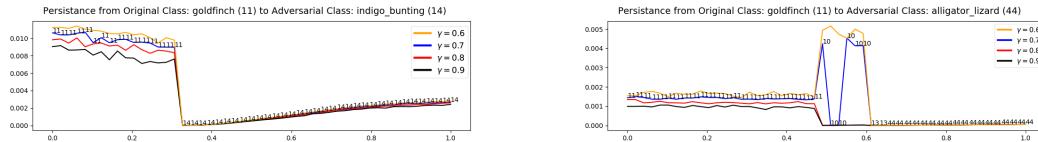


FIGURE B.6: The  $\gamma$ -persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) (left) and to an adversarial image generated with PGL in the class `alligator_lizard` (44) (right) on a vgg16 classifier with different values of  $\gamma$ . 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  $\gamma$ -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  $I\!\!R^n$ . For  $\varepsilon < \sqrt{n}$  and for  $\delta < r$  we find the following:*

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

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

*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  $\sigma$  and radius  $r$ , respectively.  $\square$

The implication is that if we fix the dimension and let  $\sigma$  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).

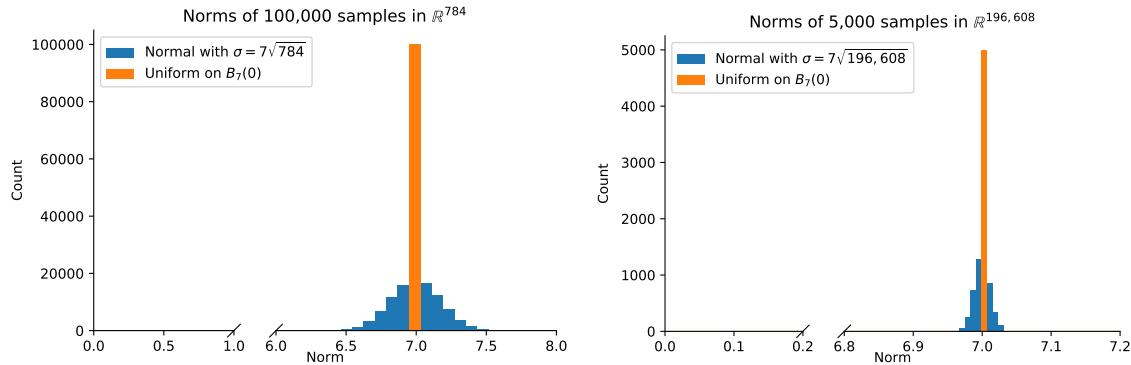


FIGURE B.7: Comparison of the length of samples drawn from  $U(B_7(0))$  and  $N(0, 7\sqrt{n})$  for  $n = 784$ , the dimension of MNIST, (left) and  $n = 196,608$ , the dimension of ImageNet, (right).

## 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

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

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

$$(\text{C.5})$$

If  $y_i^k = 0$  then we have

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

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

$$(\text{C.8})$$

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:

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

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

This formulates a kernel machine where

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

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

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

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

$$b = 0 \quad (\text{C.15})$$

□

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.

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

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

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

In order for

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

to be a kernel on its own, we need it to be a positive (or negative) definite operator and symmetric. 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:

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

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

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

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

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

$$(\text{C.24})$$

Otherwise, we get the usual

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

$$(\text{C.26})$$

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)  $K$ :

$$\text{Case 1: } L : \mathbb{R}^K \rightarrow \mathbb{R} \quad (\text{C.27})$$

$$\text{Case 2: } L : \mathbb{R}^K \rightarrow \mathbb{R}^K \quad (\text{C.28})$$

$$(\text{C.29})$$

#### Case 1 Scalar Loss

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

Let  $f$  be a vector valued function so that  $f : \mathbb{R}^D \rightarrow \mathbb{R}^K$  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  $\frac{\partial f}{\partial t}$  is a column and has shape  $K \times 1$  and our first chain rule can be done the old

fashioned way on each row of that column:

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

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

$$\text{Apply chain rule} \quad (\text{C.32})$$

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

$$\text{Let} \quad (\text{C.34})$$

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

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

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

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

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

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

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

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

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

$$(\text{C.42})$$

Note, now that we are summing over  $j$ , so we can write this as an inner product on  $j$  with the  $\nabla$  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

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

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

$$(\text{C.45})$$

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  $X$ :

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

This additional dependence of  $\varepsilon$  on  $s$  simply forces us to keep  $\varepsilon$  inside the summation in equation ??.

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

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

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

This additional dependence of  $\varepsilon$  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 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

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

$$\text{Var}(\bar{\mu}(X') - \mu(X')) = [K(X', X')] - [K(X', X)] [K(X, X)]^{-1} [K(X, X')] \quad (\text{C.50})$$

$$(\text{C.51})$$

Where

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

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  $\text{tr}(K(A_i, B_i))$  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](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](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=Skth1LkPf>.
- 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](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=Hkg4TI9x1>.
- Huang, Rui, Andrew Geng, and Yixuan Li (2021a). “On the Importance of Gradients for Detecting Distributional Shifts in the Wild”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc’Aurelio Ranzato et al., pp. 677–689. URL: <https://proceedings.neurips.cc/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html>.

- (2021b). “On the Importance of Gradients for Detecting Distributional Shifts in the Wild”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc’Aurelio Ranzato et al., pp. 677–689. URL: <https://proceedings.neurips.cc/paper/2021/hash/063e26c670d07bb7c4d30e6fc69fe056-Abstract.html>.
- Huang, Rui and Yixuan Li (2021). “MOS: Towards Scaling Out-of-Distribution Detection for Large Semantic Space”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, pp. 8710–8719. DOI: 10.1109/CVPR46437.2021.00860. URL: <https://openaccess.thecvf.com/content/CVPR2021/html/Huang\_MOS\_Towards\_Scaling\_Out-of-Distribution\_Detection\_for\_Large\_Semantic\_Space\_CVPR\_2021\_paper.html>.
- 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.
- 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](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://doi.org/10.1109/CVPR46437.2021.01506>.

- //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/01894d6f048493d2cacde3c579c3>
- 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](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: 34th 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/01894d6f048493d2cacde3c579c3>

- 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/f3b7e5d3eb074cde5b76e26bc0fb5776-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](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.