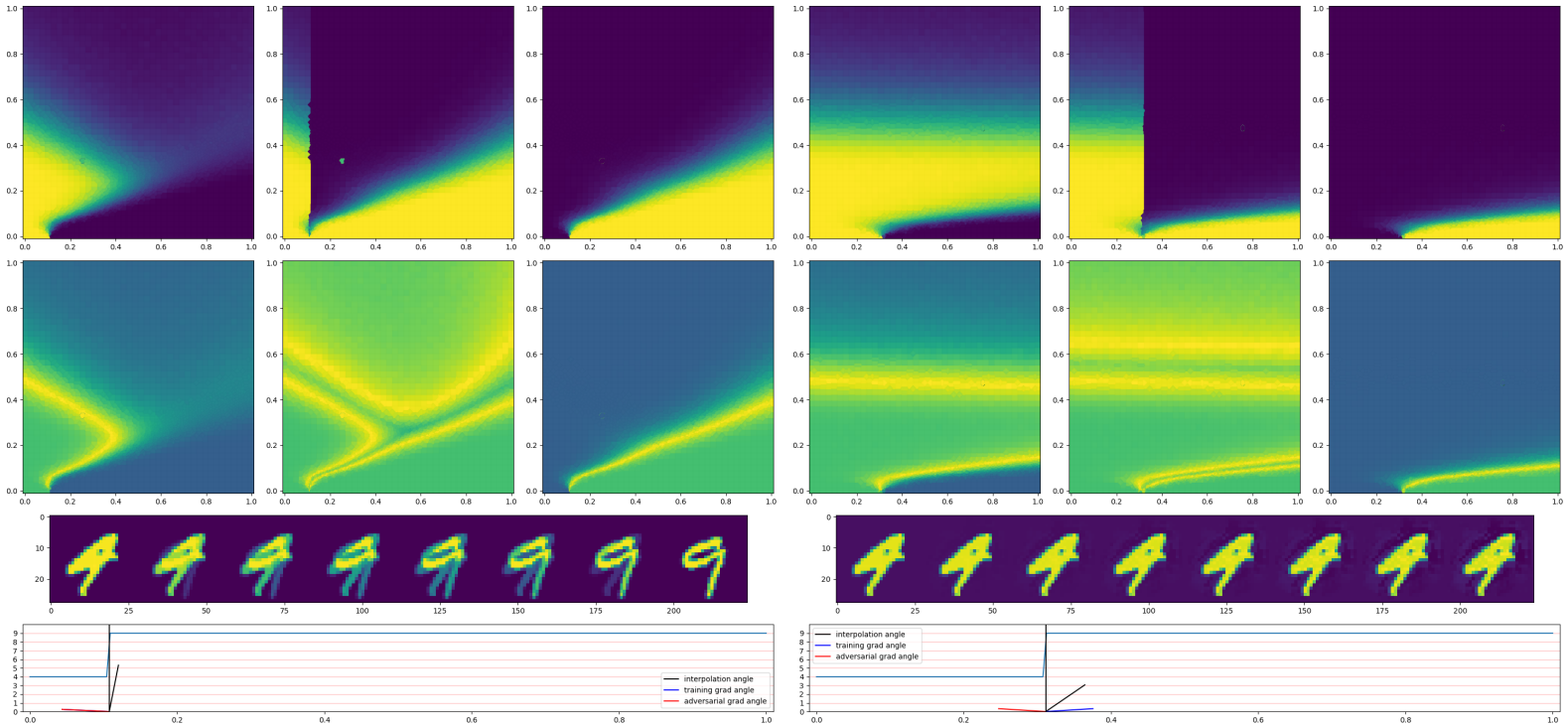
A Geometric Framework for Adversarial Vulnerability in Machine Learning



Brian Bell : University of Arizona

November 7, 2023

Contents

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.
- Define a geometric approach to interpreting neural network classifiers.

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.
- Define a geometric approach to interpreting neural network classifiers.
- Connect geometric approach with concept of robustness.

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.
- Define a geometric approach to interpreting neural network classifiers.
- Connect geometric approach with concept of robustness.
- Define a kernel based representation which allows application of kernel based tools to ANNs

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.
- Define a geometric approach to interpreting neural network classifiers.
- Connect geometric approach with concept of robustness.
- Define a kernel based representation which allows application of kernel based tools to ANNs
- Leverage our kernel based representation and these tools to get some useful results!

Goals

- Understand (mathematically) what Artificial Neural Networks (ANNs) are and how they are being used.
- Define a geometric approach to interpreting neural network classifiers.
- Connect geometric approach with concept of robustness.
- Define a kernel based representation which allows application of kernel based tools to ANNs
- Leverage our kernel based representation and these tools to get some useful results!
- Lay out further work which will connect representation approach to the geometric properties observed earlier.

Contents

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).
- Perceptrons are assembled into multilevel (deep) networks $(A_n \circ f_n \circ \cdots \circ f_3 \circ A_2 \circ f_2 \circ A_1 \cdots f_1)$ by Ivakhnenko and Lapa (1965)

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).
- Perceptrons are assembled into multilevel (deep) networks $(A_n \circ f_n \circ \cdots \circ f_3 \circ A_2 \circ f_2 \circ A_1 \cdots f_1)$ by Ivakhnenko and Lapa (1965)

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).
- Perceptrons are assembled into multilevel (deep) networks $(A_n \circ f_n \circ \cdots \circ f_3 \circ A_2 \circ f_2 \circ A_1 \cdots f_1)$ by Ivakhnenko and Lapa (1965)
- Minsky and Papert (1969) present a proof that basic perceptrons could not encode exclusive-or.

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).
- Perceptrons are assembled into multilevel (deep) networks $(A_n \circ f_n \circ \cdots \circ f_3 \circ A_2 \circ f_2 \circ A_1 \cdots f_1)$ by Ivakhnenko and Lapa (1965)
- Minsky and Papert (1969) present a proof that basic perceptrons could not encode exclusive-or.
- Neural Networks become disassociated from Cognitive Science and Computational Limitations curtail industrial applications.

History : Beginnings in Theory of Cognition

- The mechanics of cognition are described in the context of computation by McCulloch and Pitts (1943).
- The perceptron $f(x) = A(w \cdot x + b)$, the most granular element of a neural network, is proposed by Rosenblatt (1958).
- Perceptrons are assembled into multilevel (deep) networks $(A_n \circ f_n \circ \cdots \circ f_3 \circ A_2 \circ f_2 \circ A_1 \cdots f_1)$ by Ivakhnenko and Lapa (1965)
- Minsky and Papert (1969) present a proof that basic perceptrons could not encode exclusive-or.
- Neural Networks become disassociated from Cognitive Science and Computational Limitations curtail industrial applications.
- Interest in Neural Networks wanes.

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!
- A Harvard student, (Werbos, 1974), applies this technique to ANNs.

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!
- A Harvard student, (Werbos, 1974), applies this technique to ANNs.
- McClelland et al. (1986) propose distributed processing in the context of cognition allowing tree-like computations to be processed in separate computing threads.

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!
- A Harvard student, (Werbos, 1974), applies this technique to ANNs.
- McClelland et al. (1986) propose distributed processing in the context of cognition allowing tree-like computations to be processed in separate computing threads.
- Finally, these pieces are brought together by LeCun et al. (1989).

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!
- A Harvard student, (Werbos, 1974), applies this technique to ANNs.
- McClelland et al. (1986) propose distributed processing in the context of cognition allowing tree-like computations to be processed in separate computing threads.
- Finally, these pieces are brought together by LeCun et al. (1989).
- (LeCun et al., 1995) invent convolutional neural networks which are more capable and scale more cheaply.

History : Revolution

- Linnainmaa (1970) proposes a computation for gradients of large-scale multi-parameter models in his masters thesis. Back-Propagation is born!
- A Harvard student, (Werbos, 1974), applies this technique to ANNs.
- McClelland et al. (1986) propose distributed processing in the context of cognition allowing tree-like computations to be processed in separate computing threads.
- Finally, these pieces are brought together by LeCun et al. (1989).
- (LeCun et al., 1995) invent convolutional neural networks which are more capable and scale more cheaply.
- This completed tools is applied to the lucrative task of handwriting recognition (LeCun et al., 1998) and the commercial viability of ANNs is established.

History : Explosion

- Neural Networks' industrial success fuels a new wave of serious research and development including the now famous PyTorch (Collobert et al., 2002)

History : Explosion

- Neural Networks' industrial success fuels a new wave of serious research and development including the now famous PyTorch (Collobert et al., 2002)
- Geoffrey Hinton is among the first to comprehensively understand how to scale "deep" learning models (Hinton and Salakhutdinov, 2006; Hinton et al., 2006) along with Samy Bengio (Bengio et al., 2009).

History : Explosion

- Neural Networks' industrial success fuels a new wave of serious research and development including the now famous PyTorch (Collobert et al., 2002)
- Geoffrey Hinton is among the first to comprehensively understand how to scale "deep" learning models (Hinton and Salakhutdinov, 2006; Hinton et al., 2006) along with Samy Bengio (Bengio et al., 2009).
- Recurrent ANNs are applied to natural language processing (Collobert et al., 2011)

History : Explosion

- Neural Networks' industrial success fuels a new wave of serious research and development including the now famous PyTorch (Collobert et al., 2002)
- Geoffrey Hinton is among the first to comprehensively understand how to scale "deep" learning models (Hinton and Salakhutdinov, 2006; Hinton et al., 2006) along with Samy Bengio (Bengio et al., 2009).
- Recurrent ANNs are applied to natural language processing (Collobert et al., 2011)
- Szegedy et al. (2014) discover easily scalable adversarial attacks against neural networks!

History : Explosion

- Neural Networks' industrial success fuels a new wave of serious research and development including the now famous PyTorch (Collobert et al., 2002)
- Geoffrey Hinton is among the first to comprehensively understand how to scale "deep" learning models (Hinton and Salakhutdinov, 2006; Hinton et al., 2006) along with Samy Bengio (Bengio et al., 2009).
- Recurrent ANNs are applied to natural language processing (Collobert et al., 2011)
- Szegedy et al. (2014) discover easily scalable adversarial attacks against neural networks!
- Networks trained on Google's ImageNet database (>14M images, >20K categories) outperform humans on classification tasks

Definitions : Building Blocks

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

Definition

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

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

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

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

Definitions : Perceptron

Perceptrons have a few notable properties

- $w \cdot x + b$ is linear.
- The activation function f must contain all non-linearity needed for universal function approximation.
- In order to approximate arbitrary nonlinear functions, f must not be linear (Attali and Pagès, 1997).
- Arbitrarily many perceptrons can be connected in a tree-structure.

Definitions : ReLU

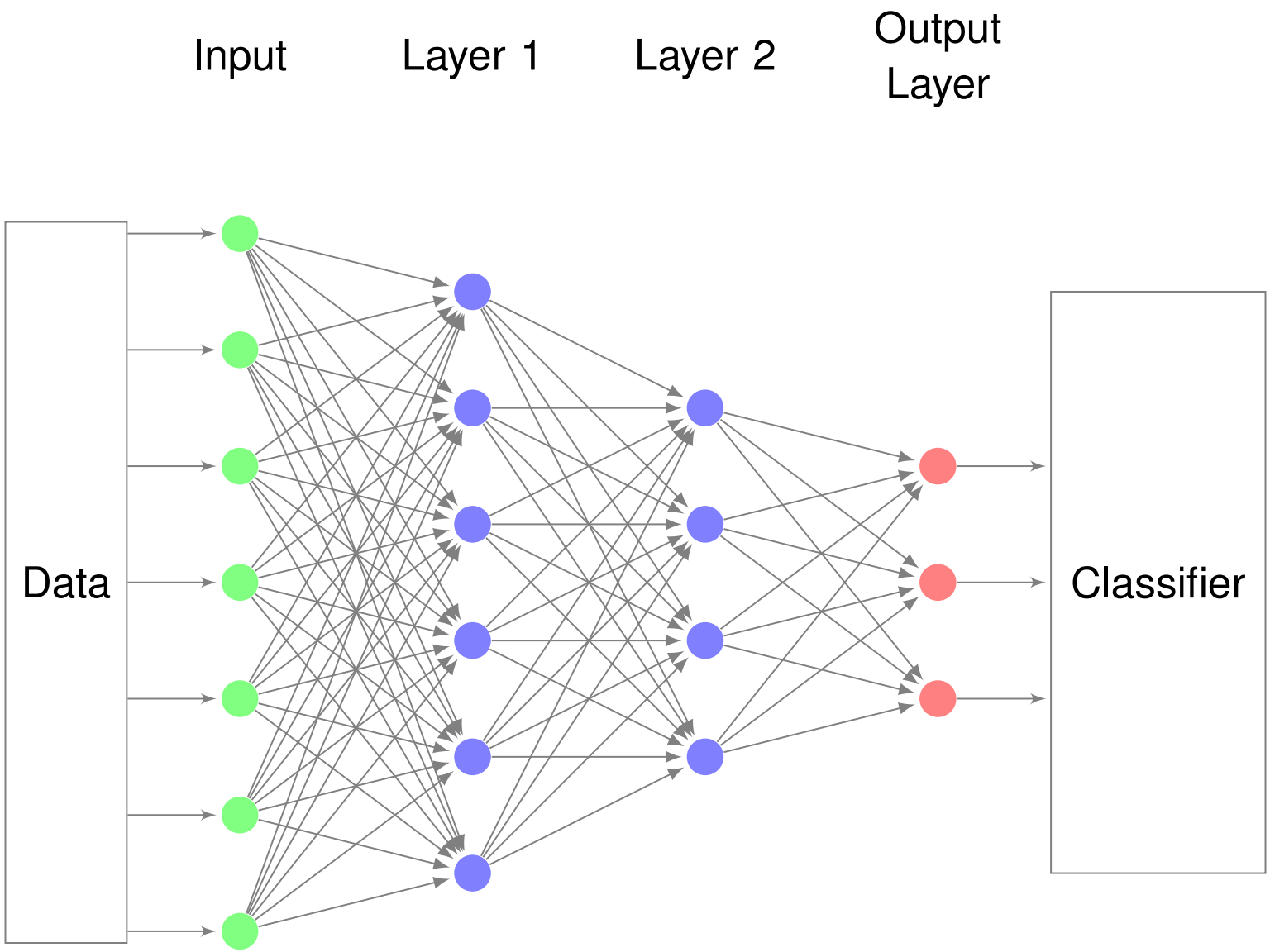
Definition

The Rectified Linear Unit (ReLU) function is

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

- Glorot et al. (2011) showed that Rectified Linear Units (ReLU) can out-perform smoother activation functions e.g. sigmoids.
- ReLU Networks even converge faster according to Nair and Hinton (2010)!
- Petersen and Voigtlaender (2018) demonstrated that this single nonlinearity of this activation function at $x = 0$ is sufficient to guarantee existence of ϵ approximation of smooth functions

Example : Fully Connected Feed Forward Network



Definition : Softmax Classifier

Definition

Softmax (or the normalized exponential) is the function given by

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

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

Definition

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

$$\text{(Output Classification) } c_s(\vec{x}) = \text{argmax}_i s_i(\vec{x})$$

Example : Other Network Structures

- **Convolutional Neural Networks** (CNNs) arrange nodes spatially and convolve kernels across this spatially producing output with multiple channels (corresponding with each individual kernel used) and preserving spatial adjacency.
- **Recurrent Neural Networks** (RNNs) admit prior network activations as inputs during computation allowing limited “memory” while working on time-varying data.

Training ANNs

1. Pick a Training Set
2. Pick a loss function One commonly used loss function for classification is known as Cross-Entropy Loss:

Definition

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

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

3. Pick a first-order optimization scheme (ODE solver).

Training : Optimization

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

Definition

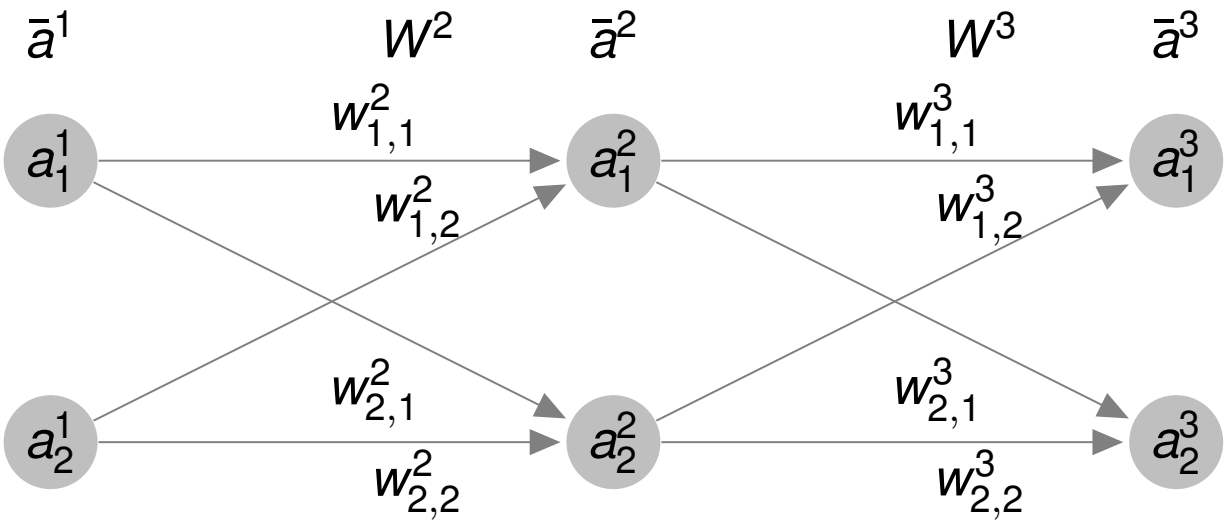
Given a loss function L, the Empirical Risk over a training dataset (X, Y) of size N is

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

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

Computation of Gradient via Backpropagation

Index: i Index: α Index: λ



Where

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

Recursively, we will define

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

Computation of Gradient via Backpropagation

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

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

Many of the terms of this gradient (e.g. the activations a_i^n and the sums $\sum_i w_{i,j}^n a_i$) are computed during forward propagation when using the network to generate output.

Computation of Gradient via Backpropagation

We can see that all of the partials will be of the form $\frac{\partial L}{\partial w^l_{n,i}} = \delta^l_n a^l_i$ where δ^l_n will contain terms which are either pre-computed or can be computed analytically. We will write this recursively in matrix form:

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

Where $\odot$ signifies element-wise multiplication.
Then we can write the gradient with respect to each layer's matrix W^l:

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

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

Optimizing Weights

Definition

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

Batch Stochastic Gradient Descent

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

Contents

Intriguing Properties of Neural Networks ?

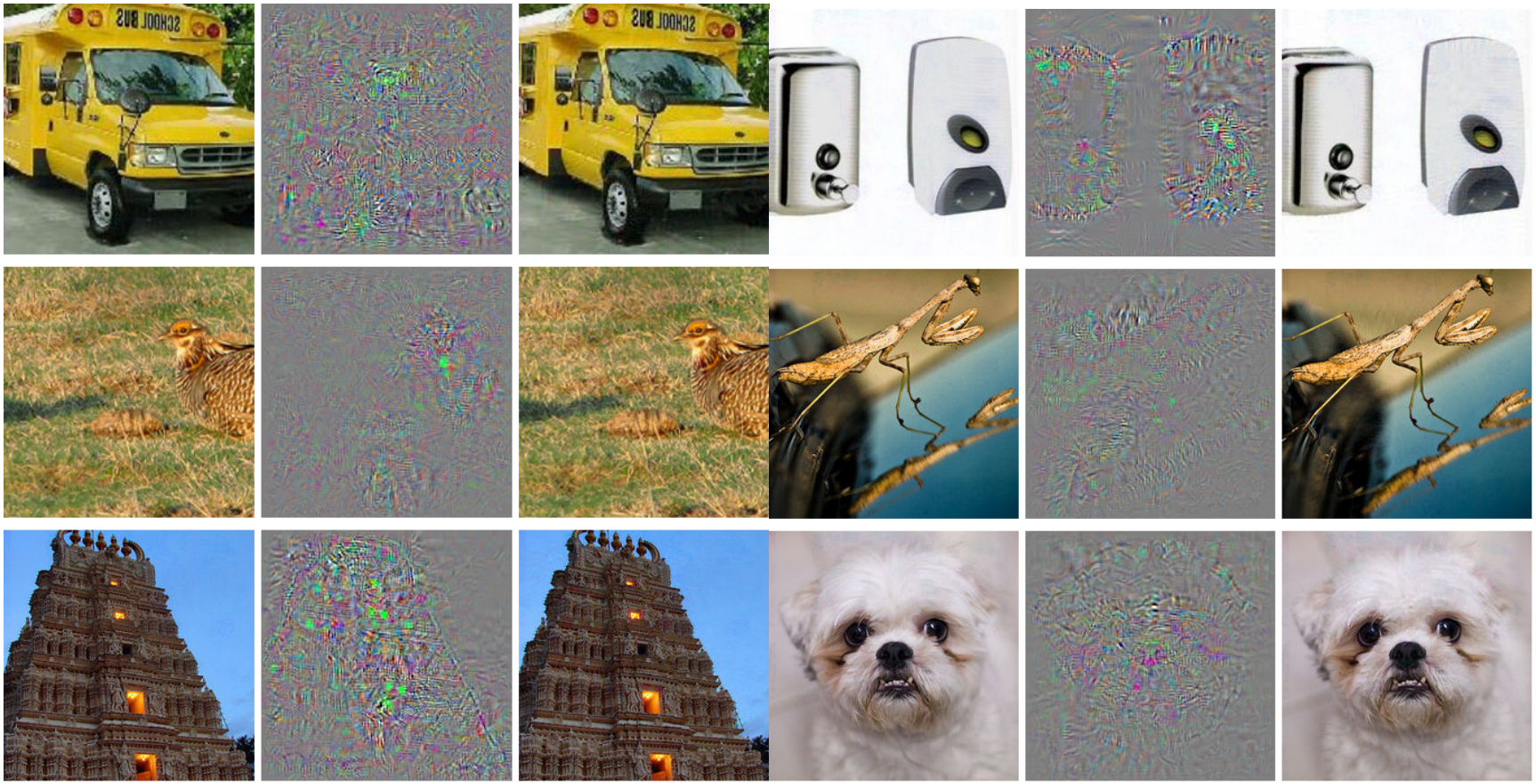


Figure: Natural Images are in columns 1 and 4, Adversarial images are in columns 3 and 6, and the difference between them (magnified by a factor of 10) is in columns 2 and 5. All images in columns 3 and 6 are classified by AlexNet as "Ostrich" ?

Attacks : L-BFGS

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

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

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

The solution is approximated with L-BFGS as implemented in Pytorch or Keras. This technique yields examples that are close to their original counterparts in the L^2 sense.

Attacks : L-BFGS : MNIST

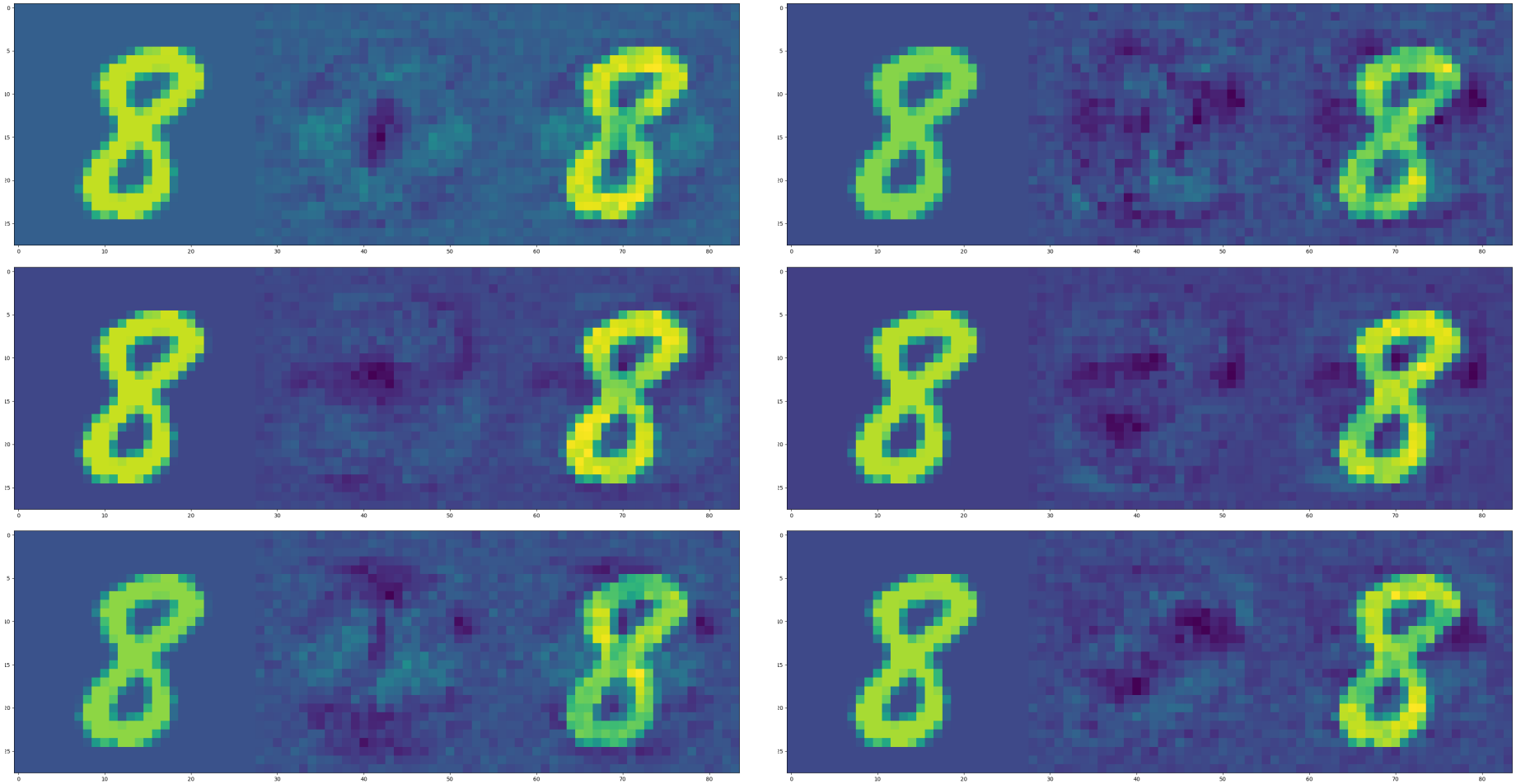


Figure: Original images on the left, Perturbation is in the middle, Adversarial Image (total of Original with Perturbation) is on the right. Column 1 shows an original 8 being perturbed to adversarial classes 0, 2, and 4. Column 2 shows adversarial classes 1, 3, and 5

Attacks : Distortion

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

Definition

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

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

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

Attacks : L-BFGS : MNIST

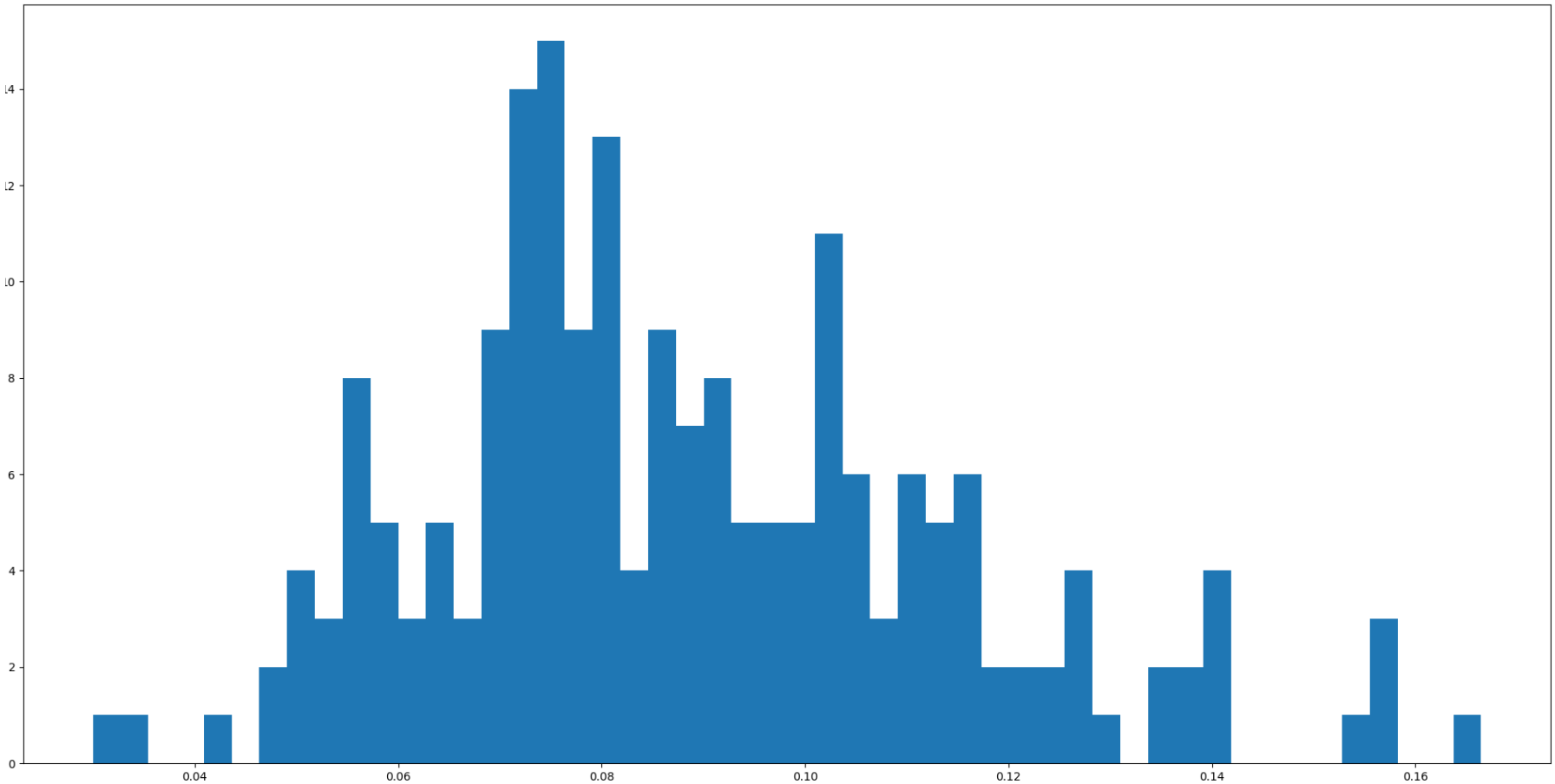


Figure: A histogram of the distortion measured for each of 900 adversarial examples generated using L-BFGS against the FC-200-200-10 network on Mnist. Mean distortion is 0.089.

Attacks : L-BFGS : ImageNet

Figure: Original images on the left, Perturbation (magnified by a factor of 100) by is in the middle, Adversarial Image (total of Original with Perturbation) is on the right. Adversarial classes are Burrito, Bison, Taxi, and Paddle Wheel (Top Left, Top Right, Bottom Left, Bottom Right)

Attacks : L-BFGS : ImageNet

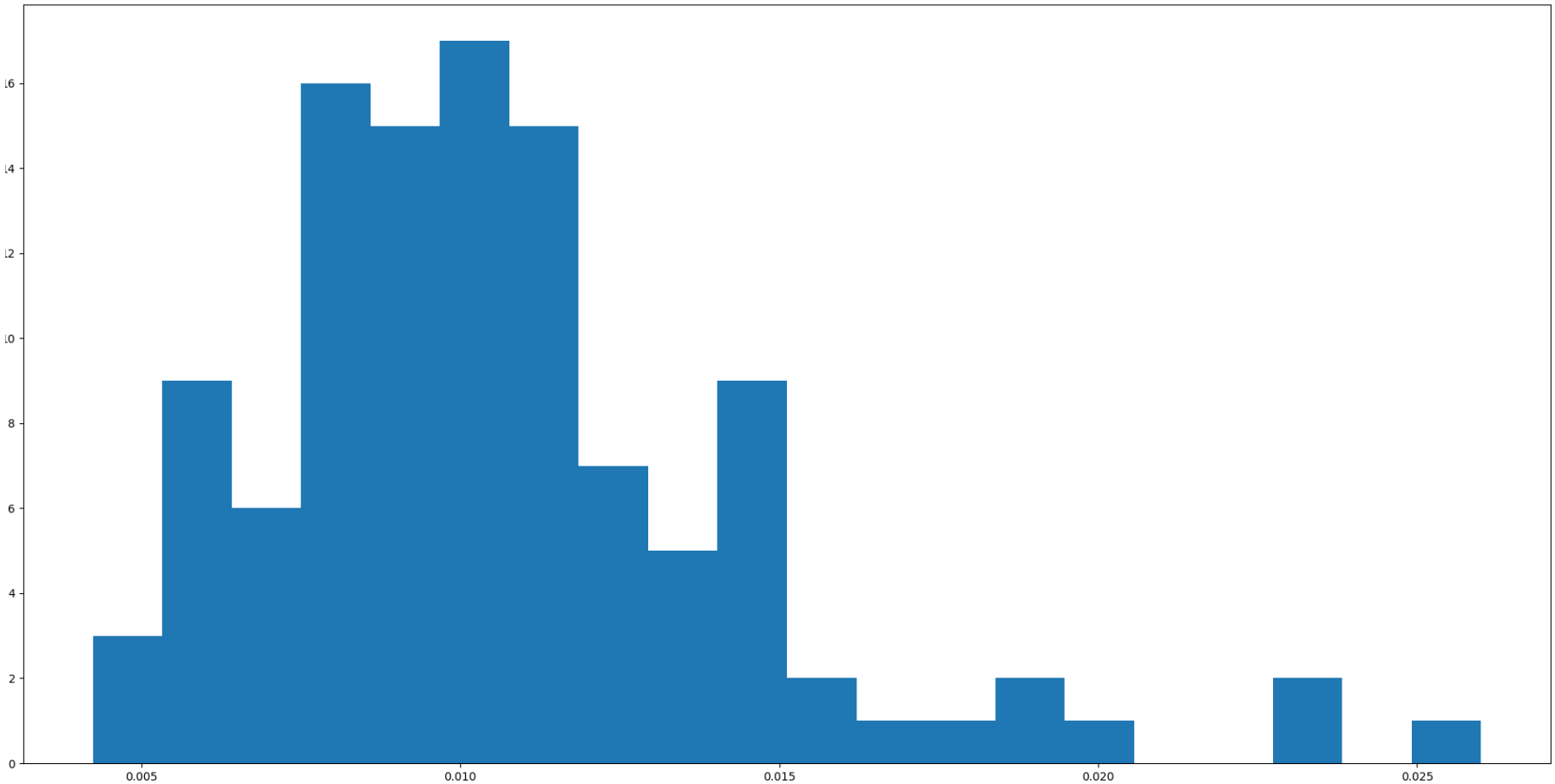


Figure: A histogram of the distortion measured for each of 112 adversarial examples generated using L-BFGS against the VGG16 network on ImageNet images with mean distortion 0.0107

Attacks : FGSM

A single step attack process using the gradient of the loss function L with respect to an image to find the adversarial perturbation Goodfellow et al. (2014). for given ε, the modified image $\hat{x}$ is computed as

$$\hat{x} = x + \epsilon \text{sign}(\nabla L(P_w(x), x)) \tag{2}$$

This method is simpler and much faster to compute than the L-BFGS technique described above, but produces adversarial examples less reliably and with generally larger distortion.

Attacks : IGSM

In Kurakin et al. (2016) an iterative application of FGSM was proposed. After each iteration, the image is clipped to a εL_∞ neighborhood of the original. Let $x'_0 = x$, then after m iterations, the adversarial image obtained is:

$$x'_{m+1} = \text{Clip}_{x,\epsilon}\Big\{ x'_m + \alpha \times \text{sign}(\nabla \ell(F(x'_m), x'_m)) \Big\} \tag{3}$$

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

Attacks : IGSM : ImageNet

Figure: adversarial example generated against VGG16 (ImageNet) with IGSM. Original Image (Rose Hip) on the left, adversarial image (Baseball) on the right.

Defining Adversarial Examples

Definition

Consider a point $x \in X$ with corresponding class $c \in C$ and a classifier $\mathcal{C} : X \to C$. We say that x admits an $(\varepsilon, d)-$*adversarial example* on $\mathcal{C}$ if there exists a point $\hat{x}$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) \neq c$.

This definition refers to the most general case of intentional mis-classification. The adversarial class can also be explicitly targeted:

Definition

Consider a point $x \in X$ with corresponding class $c \in C$ and a classifier $\mathcal{C} : X \to C$. We say that x admits an $(\varepsilon, d, c_t)-$*targeted adversarial example* on $\mathcal{C}$ if there exists a point $\hat{x}$ such that $d(x, \hat{x}) < \varepsilon$ and $\mathcal{C}(\hat{x}) = c_t$.

Contents

Background

Adversarial examples are not just a peculiarity, but seem to occur for most, if not all, DNN classifiers. For example, Shafahi et al. (2018) used isoperimetric inequalities on high dimensional spheres and hypercubes to conclude that there is a reasonably high probability that a correctly classified data point has a nearby adversarial example.
This has been reiterated using mixed integer linear programs to rigorously check minimum distances necessary to achieve adversarial conditions (Tjeng et al., 2017). Ilyas et al. (2019) showed that adversarial examples can arise from features that are good for classification but not robust to perturbation.

Justification

We will develop a statistic extending prior work on smoothed classifiers by Cohen et al. (2019).
We will be checking some geometric properties related to the alignment of gradients with human perception (Ganz et al., 2022; Kaur et al., 2019; Shah et al., 2021) and with the related underlying manifold (Kaur et al., 2019; Ilyas et al., 2019) which may imply robustness.

Justifications

The main hypothesis we seek to investigate is proposed by Shamir et al. (2021) and defines smooth manifold-like structures along which model gradients with respect to input must vary. For our purposes Manifold Aligned Gradients (MAG) will refer to the property that the gradients of a model with respect to model inputs follow a given data manifold $\mathcal{M}$.
We believe these geometric properties are related to why smoothing methods have been useful in robustness tasks (Cohen et al., 2019; Lecuyer et al., 2019; Li et al., 2019). We propose three approaches in order to connect robustness with geometric properties of the decision boundary learned by ANNs:

Contributions

1. We propose and implement two metrics based on the success of smoothed classification techniques: (γ, σ)-stability and γ-persistence defined with reference to a classifier and a given point (which can be either a natural or adversarial image, for example) and demonstrate their validity for analyzing adversarial examples.
2. We interpolate across decision boundaries using our persistence metric to demonstrate an inconsistency at the crossing of a decision boundary when interpolating from natural to adversarial examples.
3. We demonstrate via direct interpolation across decision boundaries and measurement of angles of interpolating vectors relative to the decision boundary itself that dimensionality is not solely responsible for geometric vulnerability of neural networks to adversarial attack.

Related Work : Distance-based robustness

- Khoury and Hadfield-Menell (2018) define a classifier to be robust if the class of each point in the data manifold is contained in a sufficiently large ball that is entirely contained in the same class. Larger minimum radius of this cover means more robust.
- Robustness by measuring distances from the data manifold to the decision boundary (Wang et al., 2020; Xu et al., 2023b; He et al., 2018).
- Radii in practice can be extremely small, as evidenced by results on the prevalence of adversarial examples, e.g., work by Shafahi et al. (2018) and in evaluation of ReLU networks with mixed integer linear programming e.g., work by Tjeng et al. (2017).
- Tsipras et al. (2018) investigated robustness in terms of how small perturbations affect the the average loss of a classifier. They define robust accuracy in terms of how often an adversarially perturbed example classifies correctly.
- Gilmer et al. (2018) use the expected distance to the nearest different class (when drawing a data point from the data distribution) to capture robustness

Related Work : Curvature

- Roth et al. (2019) observe that adversarial examples often arise within cones, outside of which images are classified in the original class.
- Many theoretical models of adversarial examples, for instance the dimple model developed by Shamir (2021), have high curvature and/or sharp corners as an essential piece of why adversarial examples can exists very close to natural examples.
- Fixed distance based metrics are overly sensitive to sharp curvature and these cones. **Hypothesis** : a weaker (less strict) distance based metric may help examine the effective curvature around data.

Adversarial detection via sampling

- Hu et al. (2019) invert the adversarial attack process around a test point to ask how "easy" it is to find examples of other classes near that point using gradient descent.
- This method has been generalized with the developing of smoothed classification methods (Cohen et al., 2019; Lecuyer et al., 2019; Li et al., 2019) which at varying stages of evaluation add noise to the effect of smoothing output and identifying adversaries due to their higher sensitifity to perturbation.
- These methods suffer from significant computational complexity (Kumar et al., 2020) and have been shown to have fundamental limitations in their ability to rigorously certify robustness (Blum et al., 2020; Yang et al., 2020).
- In general, the results of Hu et al. (2019) indicate that considering samples of nearby points, which approximate the computation of integrals, is likely to be more successful than methods that consider only distance to the decision boundary.

Stability

Definition

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

In the common setting when $X = R^n$, we have

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

Note here that $\mathcal{C}^{-1}$ denotes preimage.
For the Gaussian measure, the probability above may be written more concretely as

$$\frac{1}{\left(\sqrt{2\pi}\sigma\right)^n} \int_{R^n} \mathbb{1}_{\mathcal{C}^{-1}(\mathcal{C}(x))}(x') e^{-\frac{|x-x'|^2}{2\sigma^2}} dx'. \tag{4}$$

Stability Approximation

We will conduct experiments in which we estimate this stability for fixed (γ, σ) pairs via a Monte Carlo sampling, in which case the integral (4) is approximated by taking N i.i.d. samples $x_k \sim \rho$ and computing

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

Note that this quantity converges to the integral (4) as $N \to \infty$ by the Law of Large Numbers.
The ability to adjust the quantity γ is important because it is much weaker than a notion of stability that requires a ball that stays away from the decision boundary as by Khoury and Hadfield-Menell (2018). By choosing γ closer to 1, we can require the samples to be more within the same class, and by adjusting γ to be smaller we can allow more overlap.

Persistence

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

Definition

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

The γ-persistence quantity σ_γ^* measures the stability of the neighborhood of a given x with respect to the output classification.

The Argmax Function in Multi-Class Problems

A central issue when writing classifiers is mapping from continuous outputs or probabilities to discrete sets of classes. Frequently argmax type functions are used to accomplish this mapping. To discuss decision boundaries, we must precisely define argmax and some of its properties. In practice, argmax is not strictly a function, but rather a mapping from the set of outputs or activations from another model into the power set of a discrete set of classes:

$$\text{argmax} : \mathbb{R}^k \to \mathcal{P}(C) \tag{5}$$

We cannot necessarily consider argmax to be a function in general as the singleton outputs of argmax overlap in an undefined way with other sets from the power set. However, if we restrict our domain carefully, we can identify certain properties.

Argmax Decision Boundaries

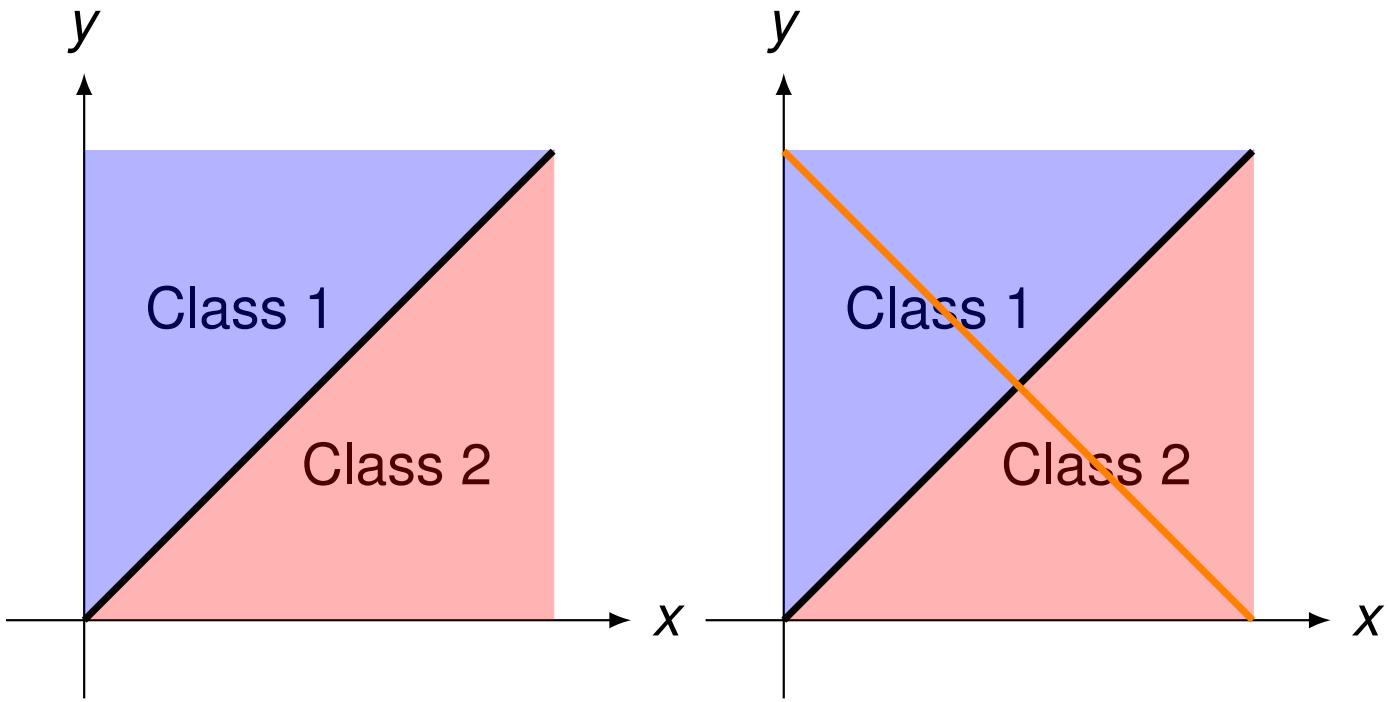


Figure: Decision boundary in $[0, 1] \times [0, 1]$ (left) and decision boundary restricted to probabilities (right) If the output of F are *probabilities* which add to one, then all points of x will map to the orange line on the right side of Figure 7.

Decision Boundary Definitions

Complement Definition

Definition

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

Level Set Definition

Definition

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

Experiments : MNIST

We begin by sampling gaussians around test points relative to a fully connected ReLU network with layers of size 784, 100, 20, and 10 and small regularization $\lambda = 10^{-7}$ which is trained on the standard MNIST training set.

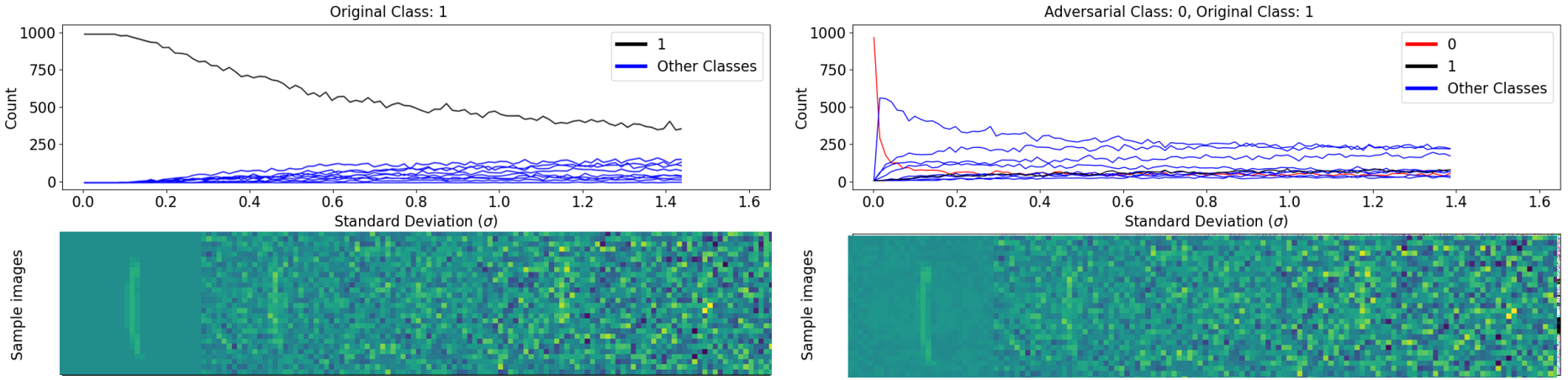


Figure: Frequency of each class in Gaussian samples with increasing variance around a natural image of class 1 (left) and around an adversarial attack of that image targeted at 0 generated using IGSM (right). The adversarial class (0) is shown as a red curve. The natural image class (1) is shown in black.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

Experiments : MNIST Persistence

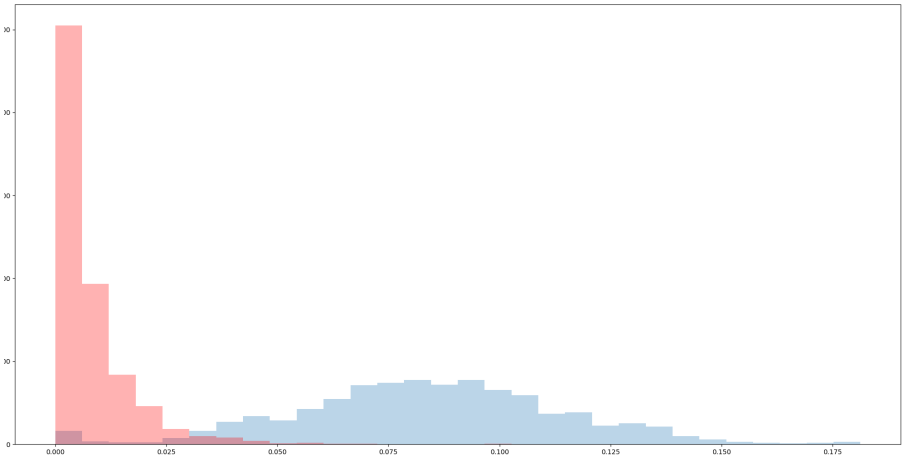


Figure: Histogram of 0.7-persistence of IGSM-based adversarial examples (red) and natural examples (blue) on MNIST.

Experiments : MNIST Persistence vs Architecture

Table: Recreation of Szegedy et al. (2013), Table 1 for the MNIST dataset. For each network, we show Testing Accuracy (in %), Average Distortion ($\|x\|_2/\sqrt{n}$) of adversarial examples, and new columns show average 0.7-persistence values for natural (Nat) and adversarial (Adv) images. 300 natural and 300 adversarial examples generated with L-BFGS were used for each aggregation.

Network	Test Acc	Avg Dist	Persist (Nat)	Persist (Adv)
FC10-4	92.09	0.123	0.93	1.68
FC10-2	90.77	0.178	1.37	4.25
FC10-0	86.89	0.278	1.92	12.22
FC100-100-10	97.31	0.086	0.65	0.56
FC200-200-10	97.61	0.087	0.73	0.56
C-2	95.94	0.09	3.33	0.027
C-4	97.36	0.12	0.35	0.027
C-8	98.50	0.11	0.43	0.0517
C-16	98.90	0.11	0.53	0.0994
C-32	98.96	0.11	0.78	0.0836

Experiments : ImageNet

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

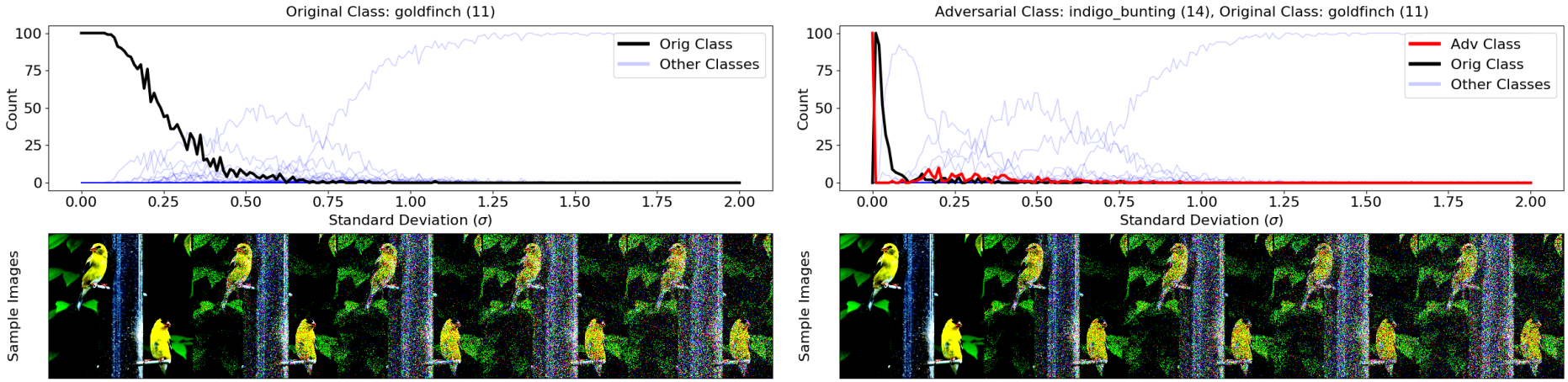


Figure: Frequency of each class in Gaussian samples with increasing variance around a `goldfinch` image (left) and an adversarial example of that image targeted at the `indigo_bunting` class and calculated using the BIM attack (right). Bottoms show example sample images at different standard deviations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

Experiments : ImageNet Persistence

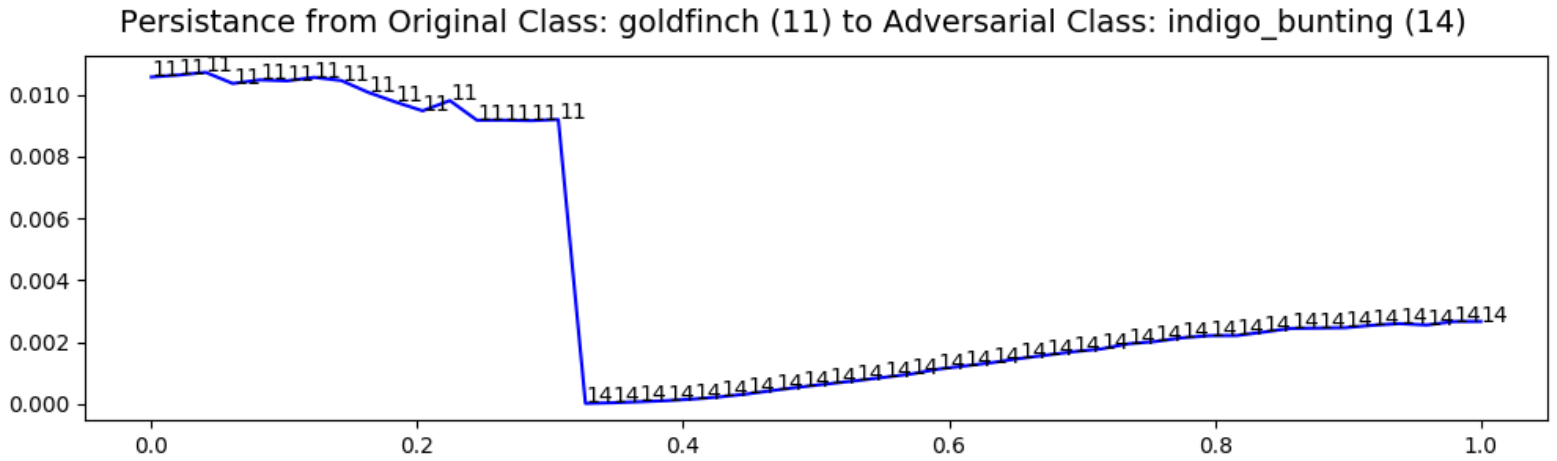


Figure: The 0.7-persistence of images along the straight line path from an image in class `goldfinch` (11) to an adversarial image generated with BIM in the class `indigo_bunting` (14) on a vgg16 classifier. The classification of each image on the straight line is listed as a number so that it is possible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

Experiments : Decision Boundary Angles

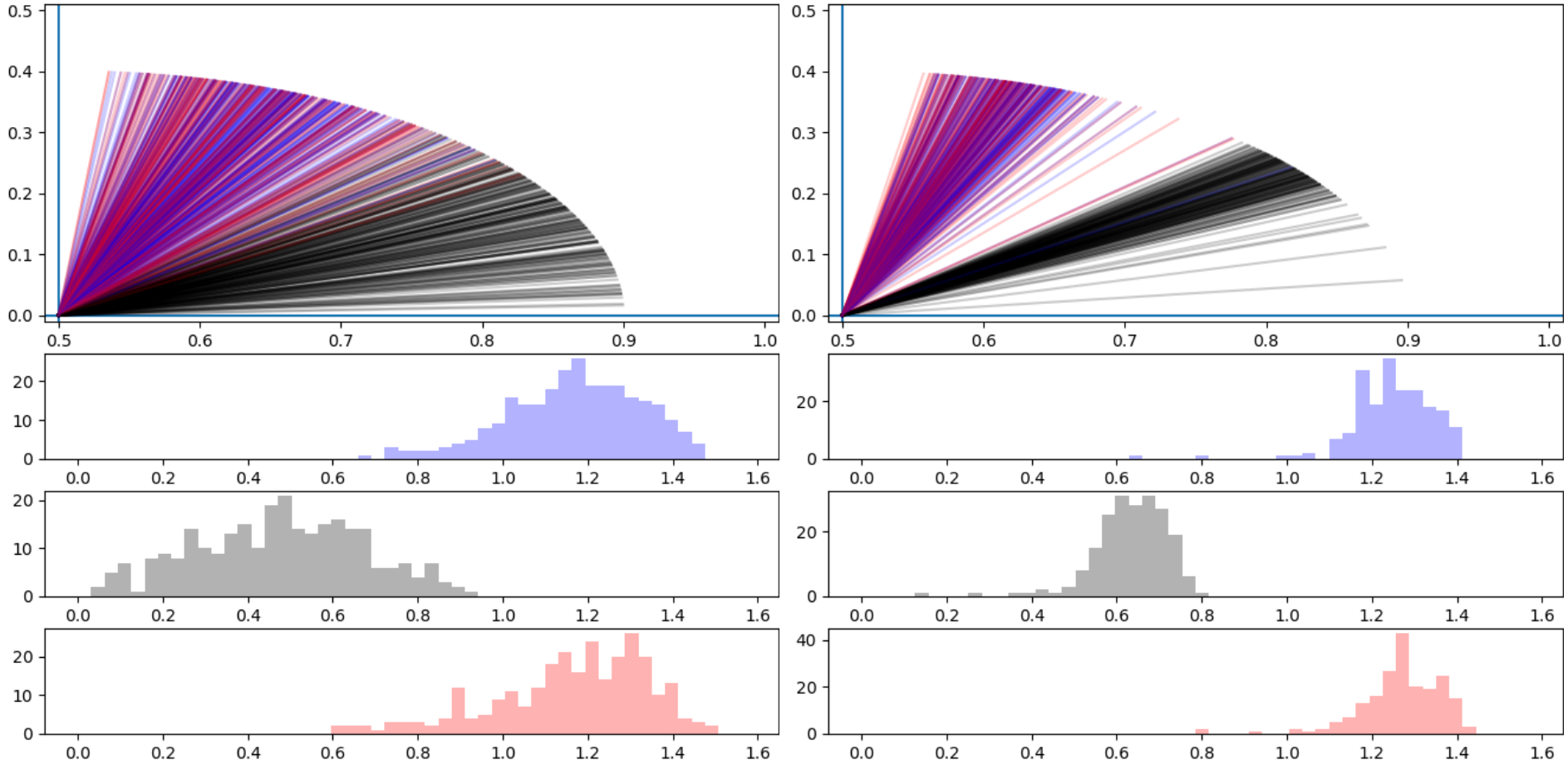


Figure: Decision boundary incident angles between test and test images (left) and between test and adversarial images (right). Angles (plotted Top) are referenced to decision boundary so $\pi/2$ radians (right limit of plots) corresponds with perfect orthogoonality to decision boundary. Lines and histograms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

Conclusions

- Geometric properties including curvature and alignment to decision boundaries seem to be related with robustness.
- Direct computation of these geometric properties is expensive and at odds with the fact that much greater scale is needed to understand these properties in practice.
- We could benefit from faster methods to analyze geometric properties from models – Monte Carlo is what you do when you can't solve your differential equations by faster means!
- Decision boundaries are askew from interpolation between natural images!

Contents

Model Optimization Step

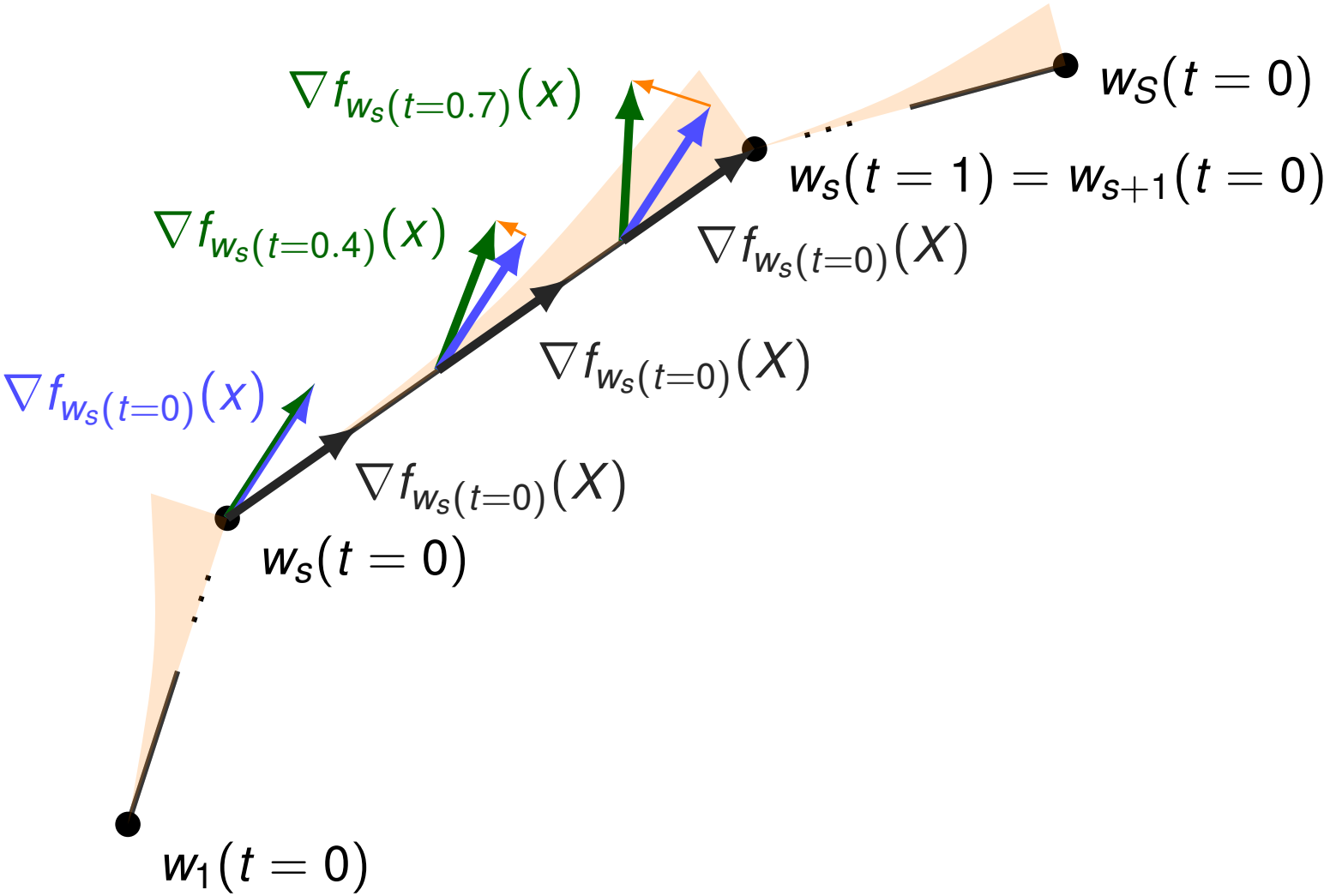


Figure: Comparison of test gradients used by Discrete Path Kernel (DPK) from prior work (Blue) and the Exact Path Kernel (EPK) proposed in this work (green) versus total training vectors (black) used for kernel formulations along a discrete

Kernel Approximation Error

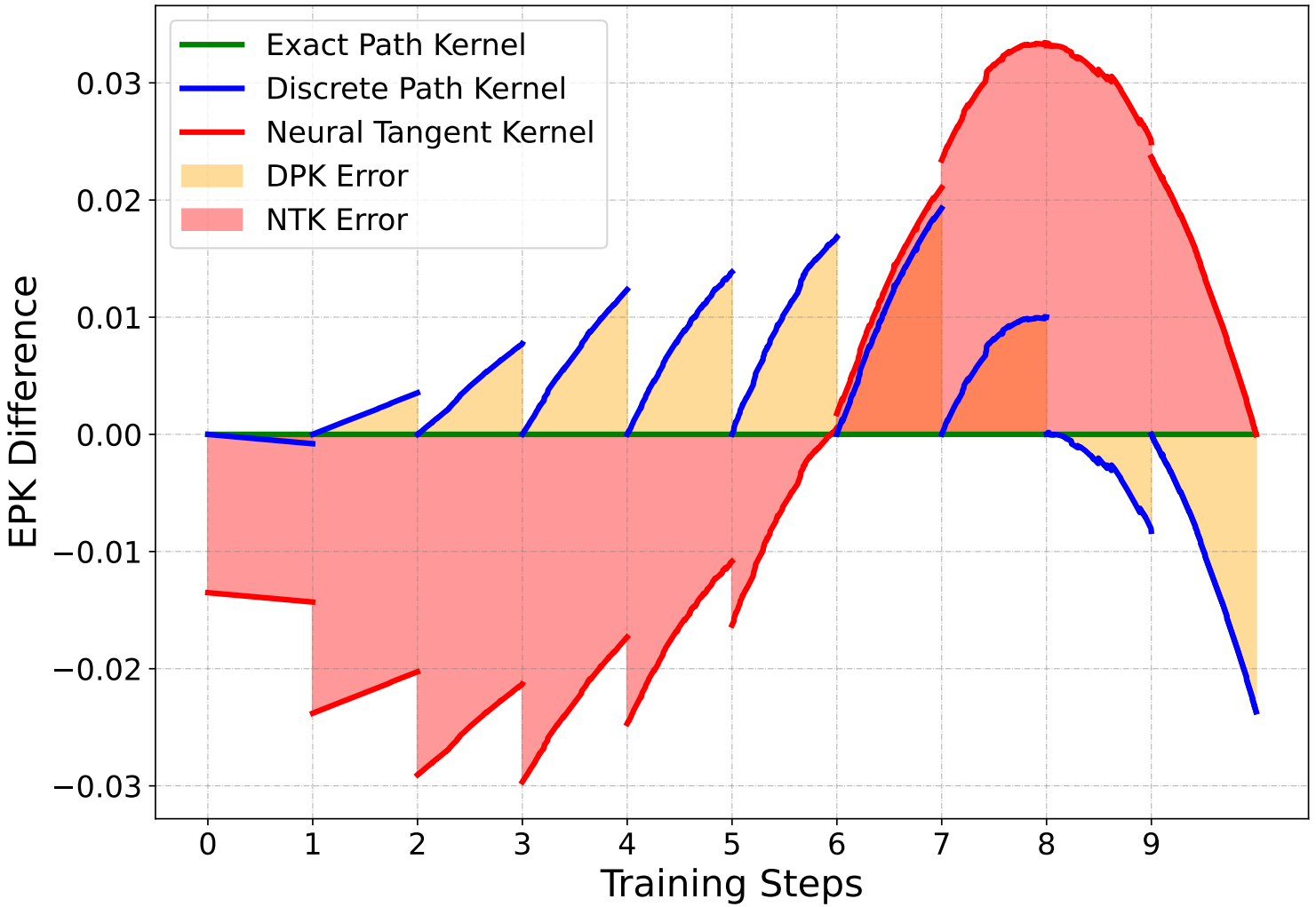


Figure: Measurement of gradient alignment on test points across the training path. The EPK is used as a frame of reference. The y-axis is exactly the difference between the EPK and other representations.

Contributions

This paper makes the following significant theoretical and experimental contributions:

1. We prove that finite-sized neural networks trained with finite-sized gradient descent steps and cross-entropy loss can be exactly represented as kernel machines using the EPK. Our derivation incorporates a previously-proposed path kernel, but extends this method to account for practical training procedures Domingos (2020a); Chen et al. (2021).
2. We demonstrate that it is computationally tractable to estimate the kernel underlying a neural network classifier, including for small convolutional computer vision models.
3. We compute Gram matrices using the EPK and use them to illuminate prior theory of neural networks and their understanding of uncertainty.
4. We employ Gaussian processes to compute the covariance of a neural network's logits and show that this reiterates previously observed shortcomings of neural network generalization.

Related Work : Neural Tangent Kernel

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

Related Work : Discrete Path Kernel

- Domingos (2020b) attempts to establish a correspondence between kernel machines and parametric models trained by gradient descent in the case of a continuous training path (i.e. the limit as gradient descent step size $\varepsilon \to 0$)
- We will refer to this continuously integrated path representation as the Discrete Path Kernel (DPK).
- Chen et al. (2021) refined this formulation to establish an equivalence, however this equivalence requires a restrictive assumptions.
- A major limitation of this formulation is its reliance on a continuous integration over a gradient flow, which differs from the discrete forward Euler steps employed in real-world model training.
- The unbounded error of this approximation limits the value of the continuous path kernel in practical scenarios (Incudini et al., 2022).

Theoretical Results

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

Theoretical Results : Kernel

Definition

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

Theoretical Results : Kernel Method Definition

Definition

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

$$\mathrm{K}(x) = b + \sum_{i \in I} a_i k(x, x_i) \tag{6}$$

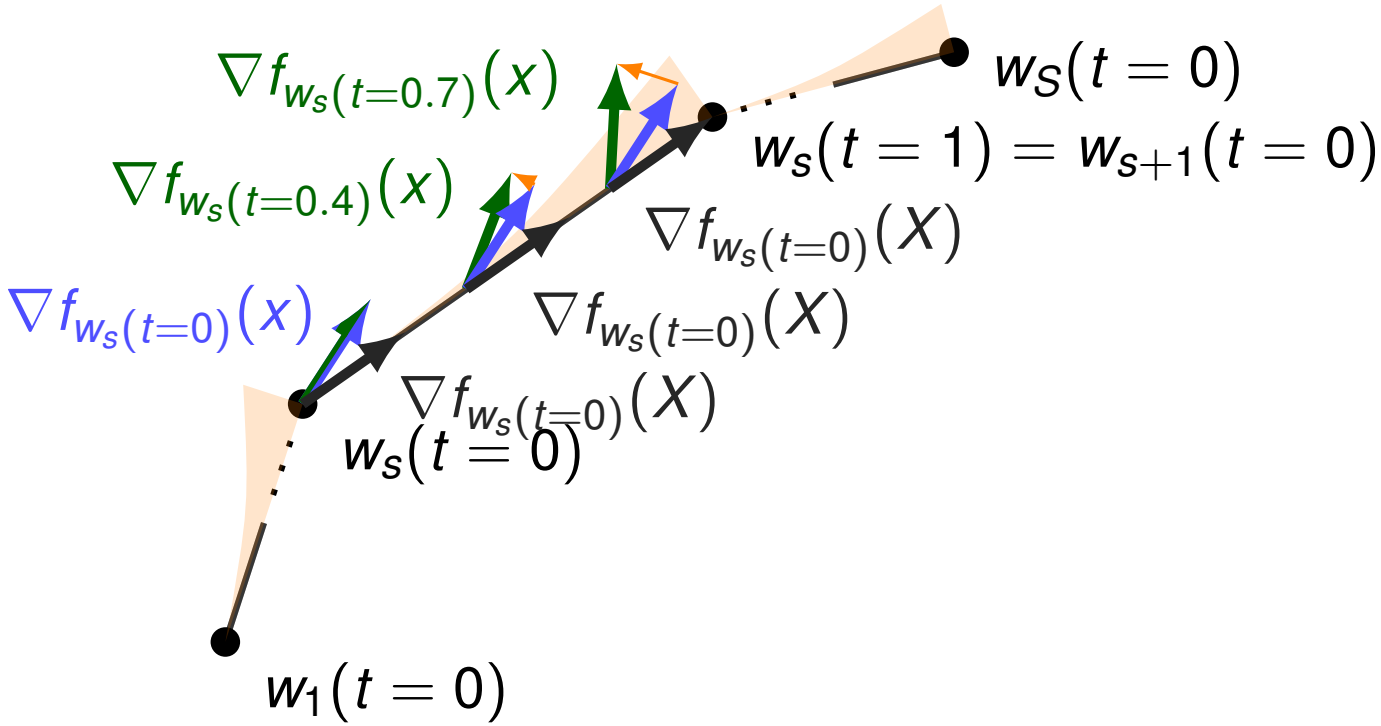
where the $a_i \in \mathbb{R}$ do not depend on x, $b \in \mathbb{R}$ is a constant, and k is a kernel. Rasmussen et al. (2006)

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

$$\mathrm{K}(x) = b + \sum_{i \in I} a_i \langle \phi(x), \phi(x_i) \rangle \tag{7}$$

Derivation

We first derive a kernel which is an exact representation of the change in model output over one training step, and then compose our final representation by summing along the finitely many steps.



Derivation

In order to compute the EPK, gradients on training data must serve two purposes. First, they are the reference points for comparison (via inner product) with test points. Second, they determine the path of the model in weight space.

We must *continuously* compare the model's gradient at test points with *fixed* training gradients along each discrete training step s whose weights we we interpolate linearly by $w_s(t) = w_s - t(w_s - w_{s+1})$. We will do this by integrating across the gradient field induced by test points, but holding each training gradient fixed along the entire discrete step taken. This creates an asymmetry, where test gradients are being measured continuously but the training gradients are being measured discretely (see Figure 13).

Derivation : Assumptions

We will redefine our data using an indicator to separate training points from all other points in the input space.

Definition

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

Exact Path Kernel : Definition

Definition

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

$$K_{\text{EPK}}(x, x', s) = \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x') \rangle dt \tag{8}$$

where

$$\phi_{s,t}(x) = \nabla_w f_{w_s(t,x)}(x) \tag{9}$$

$$w_s(t) = w_s - t(w_s - w_{s+1}) \tag{10}$$

$$w_s(t, x) = \begin{cases} w_s(0), & \text{if } x_l = 1 \\ w_s(t), & \text{if } x_l = 0 \end{cases} \tag{11}$$

Note: ϕ is deciding whether to select a continuously or discrete gradient based on whether the data is from the training or testing space.

Derivation : Exact Path Kernel

Lemma

The exact path kernel (EPK) is a kernel.

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

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

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

Derivation : Exact Path Kernel

$$\alpha^\top K_{\text{EPK}} \alpha = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \int_0^1 \langle \phi_{s,t}(x_i), \phi_{s,t}(x_j) \rangle dt \tag{12}$$

$$= \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \int_0^1 \langle \nabla_w \hat{y}_{w_s(t,x_i)}, \nabla_w \hat{y}_{w_s(t,x_j)} \rangle dt \tag{13}$$

$$= \int_0^1 \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j \langle \nabla_w \hat{y}_{w_s(t,x_i)}, \nabla_w \hat{y}_{w_s(t,x_j)} \rangle dt \tag{14}$$

$$= \int_0^1 \sum_{i=1}^{n} \sum_{j=1}^{n} \langle \alpha_i \nabla_w \hat{y}_{w_s(t,x_i)}, \alpha_j \nabla_w \hat{y}_{w_s(t,x_j)} \rangle dt \tag{15}$$

$$\tag{16}$$

Derivation : Exact Path Kernel

$$= \int_0^1 \langle \sum_{i=1}^{n} \alpha_i \nabla_w \hat{y}_{w_s(t,x_i)}, \sum_{j=1}^{n} \alpha_j \nabla_w \hat{y}_{w_s(t,x_j)} \rangle dt \tag{17}$$

Re-ordering the sums so that their indices match, we have (18)

$$= \int_0^1 \left\| \sum_{i=1}^{n} \alpha_i \nabla_w \hat{y}_{w_s(t,x_i)} \right\|^2 dt \tag{19}$$

$$\geq 0 \tag{20}$$

□ (21)

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

Derivation : Exact Kernel Ensemble Representation

Theorem (Exact Kernel Ensemble Representation)

A model f_{w_N} trained using discrete steps matching the conditions of the exact path kernel has the following exact representation as an ensemble of N kernel machines:

$$f_{w_N} = KE(x) := \sum_{s=1}^{N} \sum_{i=1}^{M} a_{i,s} K_{EPK}(x, x', s) + b \tag{22}$$

where

$$a_{i,s} = -\varepsilon \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \tag{23}$$

$$b = f_{w_0}(x) \tag{24}$$

Derivation : Exact Kernel Ensemble Representation Proof

Proof. Let f_w be a differentiable function parameterized by parameters w which is trained via N forward Euler steps of fixed step size ε on a training dataset X with labels Y, with initial parameters w_0 so that there is a constant b such that for every x, $f_{w_0}(x) = b$, and weights at each step $w_s : 0 \leq s \leq N$. Let $x \in X$ be arbitrary and within the domain of f_w for every w. For the final trained state of this model f_{w_N}, let $y = f_{w_N}(x)$.
For one step of training, we consider $y_s = f_{w_s(0)}(x)$ and $y_{s+1} = f_{w_{s+1}}(x)$.
We wish to account for the change $y_{s+1} - y_s$ in terms of a gradient flow, so we must compute $\dfrac{\partial y}{dt}$ for a continuously varying parameter t. Since f is trained using forward Euler with a step size of $\varepsilon > 0$, this derivative is determined by a step of fixed size of the weights w_s to w_{s+1}.

Derivation : Exact Kernel Ensemble Representation Proof

We parameterize this step in terms of the weights:

$$\frac{dw_s(t)}{dt} = (w_{s+1} - w_s) \tag{25}$$

$$\int \frac{dw_s(t)}{dt} dt = \int (w_{s+1} - w_s) dt \tag{26}$$

$$w_s(t) = w_s + t(w_{s+1} - w_s) \tag{27}$$

$$\tag{28}$$

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

$$\frac{dw_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(X), y_i) = -\varepsilon \sum_{j=1}^{d} \sum_{i=1}^{M} \frac{\partial L(f_{w_s(0)}(x_i), y_i)}{\partial w_j} \tag{29}$$

Derivation : Exact Kernel Ensemble Representation Proof

Applying chain rule and the above substitution, we can write

$$\frac{d\hat{y}}{dt} = \frac{df_{w_s(t)}}{dt} = \sum_{j=1}^{d} \frac{df}{\partial w_j} \frac{\partial w_j}{dt} \tag{30}$$

$$= \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \frac{\partial L(f_{w_s(0)}(X_T), Y_T)}{\partial w_j} \right) \tag{31}$$

$$= \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \frac{\partial f_{w_s(0)}(x_i)}{\partial w_j} \right) \tag{32}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \sum_{j=1}^{d} \frac{df_{w_s(t)}(x)}{\partial w_j} \frac{df_{w_s(0)}(x_i)}{\partial w_j} \tag{33}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) \tag{34}$$

Derivation : Exact Kernel Ensemble Representation Proof

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

$$y_{s+1} - y_s = \int_0^1 -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \nabla_w f_{w_s(t)}(x) \cdot \nabla_w f_{w_s(0)}(x_i) dt \quad (35)$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt \right) \cdot \nabla_w f_{w_s(0)}(x_i) \quad (36)$$

Derivation : Exact Kernel Ensemble Representation Proof

For all N training steps, we have

$$y_N = b + \sum_{s=1}^{N} y_{s+1} - y_s$$

$$y_N = b + \sum_{s=1}^{N} -\varepsilon \sum_{i=1}^{M} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \left(\int_0^1 \nabla_w f_{w_s(t)}(x) dt \right) \cdot \nabla_w f_{w_s(0)}(x_i)$$

$$= b + \sum_{i=1}^{M} \sum_{s=1}^{N} -\varepsilon \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \int_0^1 \left\langle \nabla_w f_{w_s(t,x)}(x), \nabla_w f_{w_s(t,x_i)}(x_i) \right\rangle dt$$

$$= b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} \int_0^1 \left\langle \phi_{s,t}(x), \phi_{s,t}(x_i) \right\rangle dt$$

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

Derivation : Exact Kernel Ensemble Representation

Having established this representation, we can introduce $P_S(t)$, the training path which is composed by placing each of the S training steps end-to-end. We can rewrite 37 by combining $\sum_{s=1}^{S}$ and $\int_0^1 dt$ into a single integral $\int_{P_S}$:

$$y_N = b + -\varepsilon \sum_{i=1}^{M} \int_{P_S} \frac{dL(f_{w_s(0)}(x_i), y_i)}{df_{w_s(0)}(x_i)} \left(\nabla_w f_{w_s(t)}(x)\right) \cdot \nabla_w f_{w_s(0)}(x_i) \quad (37)$$

Derivation : Exact Kernel Machine Representation

Theorem (Exact Kernel Machine Reduction)

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

$$f_{w_N}(x) = KM(x) := b + \sum_{i=1}^{M} a_{i,0} K_{NEPK}(x, x') \tag{38}$$

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

Derivation : Remarks

$\phi_{s,t}(x)$ depends on both s and t, which is non-standard but valid, however an important consequence of this mapping is that the output of this representation is not guaranteed to be continuous. This discontinuity is exactly measuring the error between the model along the exact path compared with the gradient flow for each step.
We can write another function k' which is continuous but not symmetric, yet still produces an exact representation:

$$k'(x, x') = \langle \nabla_w f_{w_s(t)}(x), \nabla_w f_{w_s(0)}(x') \rangle \tag{39}$$

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

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt = \nabla_w f_{w_s(0)}(x) \tag{40}$$

Derivation : Remarks

Since f is being trained using forward Euler, we can write:

$$\frac{\partial w_s(t)}{dt} = -\varepsilon \nabla_w L(f_{w_s(0)}(x_i), y_i) \tag{41}$$

Our parameterization of this step depends on the step size ε and as $\varepsilon \to 0$, we have

$$\int_0^1 \nabla_w f_{w_s(t)}(x) dt \approx \nabla_w f_{w_s(0)}(x) \tag{42}$$

In particular, given a model f that admits a Lipshitz constant K this approximation has error bounded by εK and a proof of this convergence is direct.

Derivation : Remarks

We can see that by changing equation 41 we can produce an exact representation under a variety of modifications:

- Any first-order optimization scheme.
- Any subset of training data (by keeping track of the exact data used to compute gradients for each step)
- Adversarial training (by keeping copies of the adversarial images used for each step)

Ensemble Reduction

In order to reduce the ensemble representation of Equation (22) to the kernel representation of Equation (45), we require that the sum over steps still retain the properties of the kernel (symmetry and positive semi-definiteness). In particular we require that for every subset of the training data x_i and arbitrary α_i and α_j, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{M}\sum_{s=1}^{N}\alpha_i\alpha_j a_{l,s}\int_0^1 K_{\text{EPK}}(x_i,x_j)dt \geq 0 \tag{43}$$

A sufficient condition for this reduction is that the gradient of the loss function does not change throughout training. This is the case for categorical cross-entropy where labels are in $\{0,1\}$. In fact, in this specific context the gradient of the loss function does not depend on $f(x)$, and are fully determined by the ground truth label, making the gradient of the cross-entropy loss a constant value throughout training.

Note on Prior Work

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

$$K(x, x_i) = \int_0^T |\nabla_f L(f_t(x_i), y_i)| \langle \nabla_w f_t(x), \nabla_w f_t(x_i) \rangle dt \tag{44}$$

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

Experimental Results

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

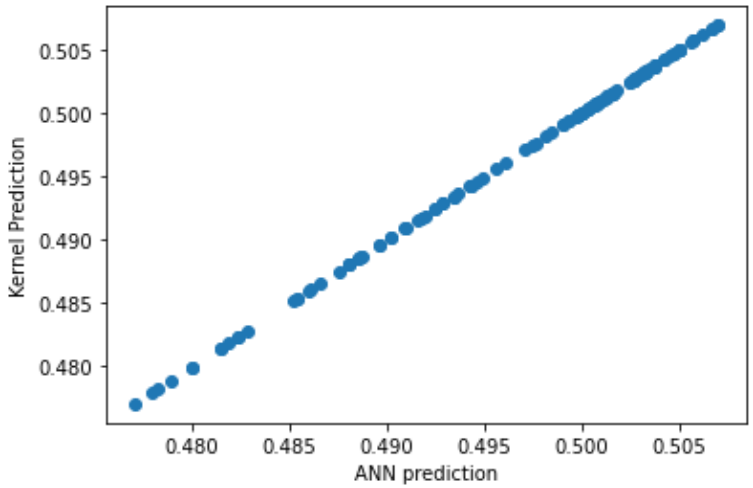
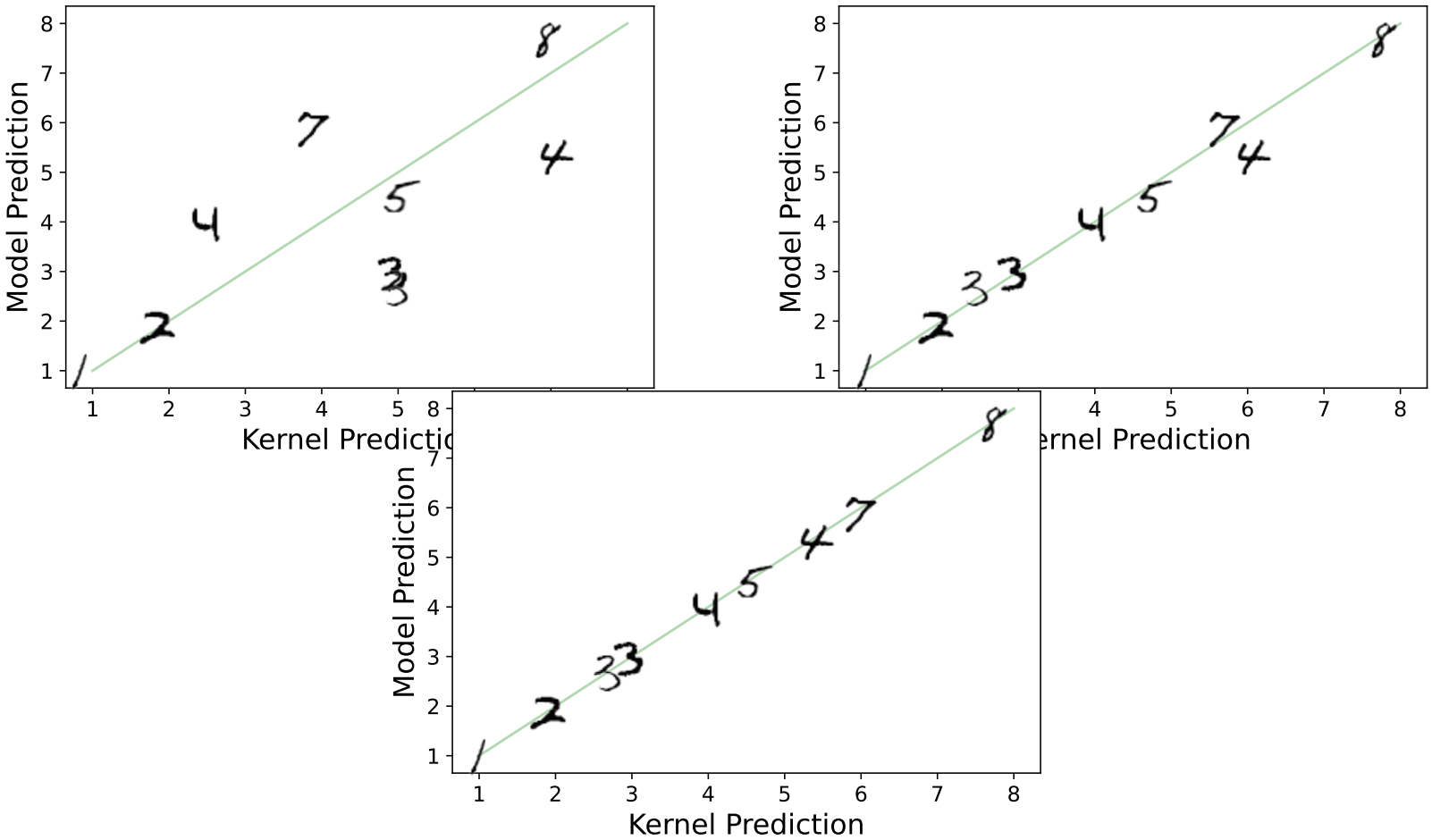


Figure: Class 1 EPK Kernel Prediction (Y) versus neural network prediction (X) for 100 test points, demonstrating extremely close agreement.

Experimental Results



Kernel Properties

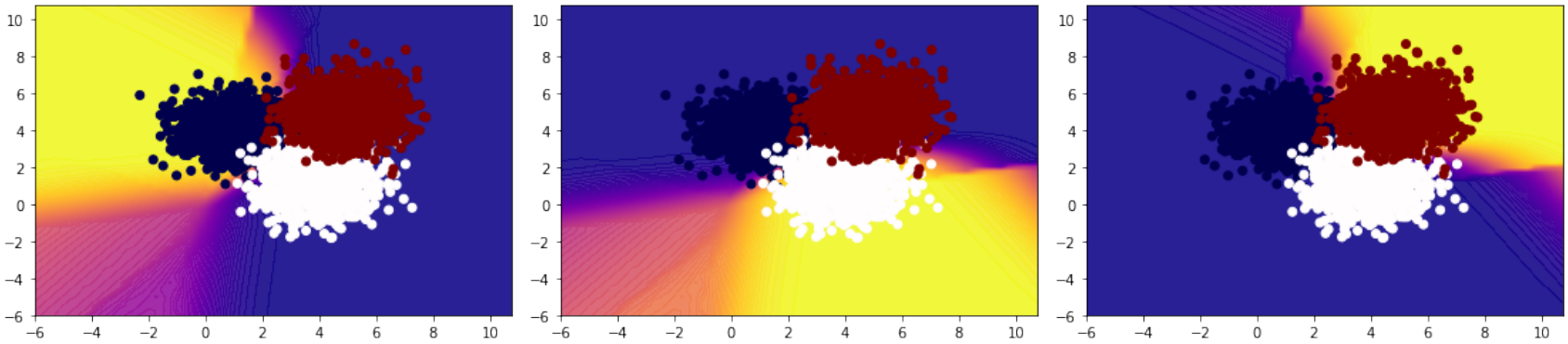
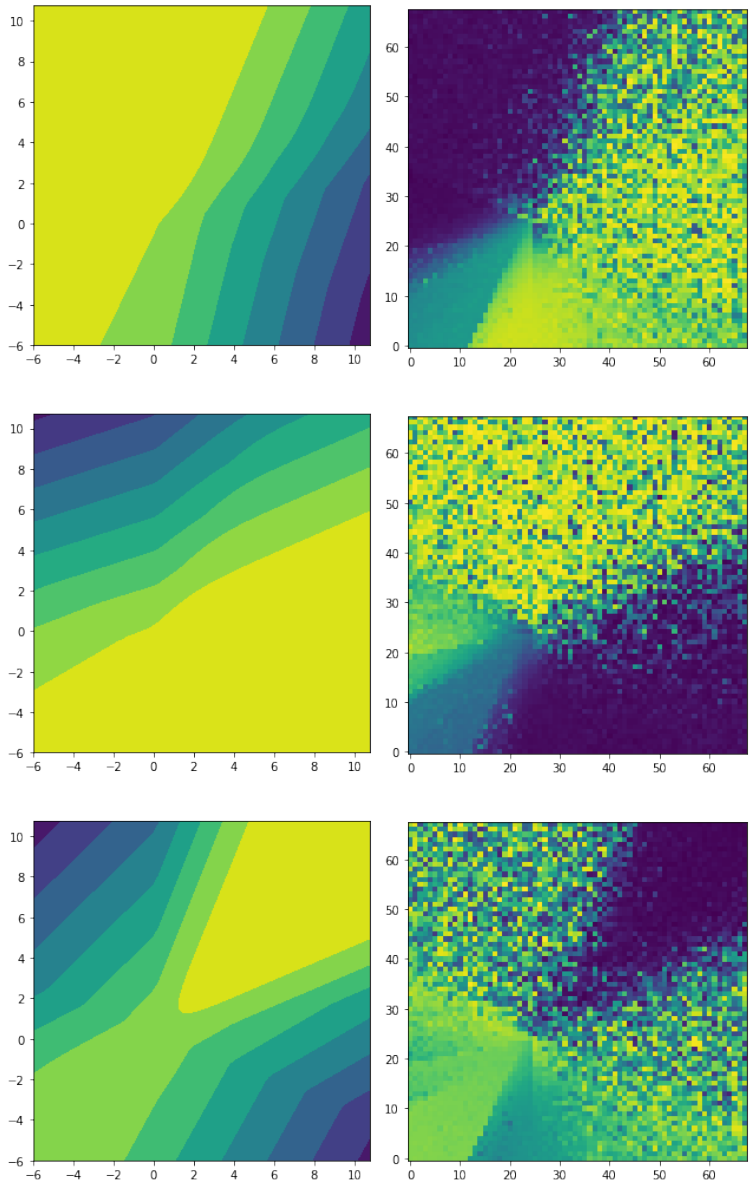
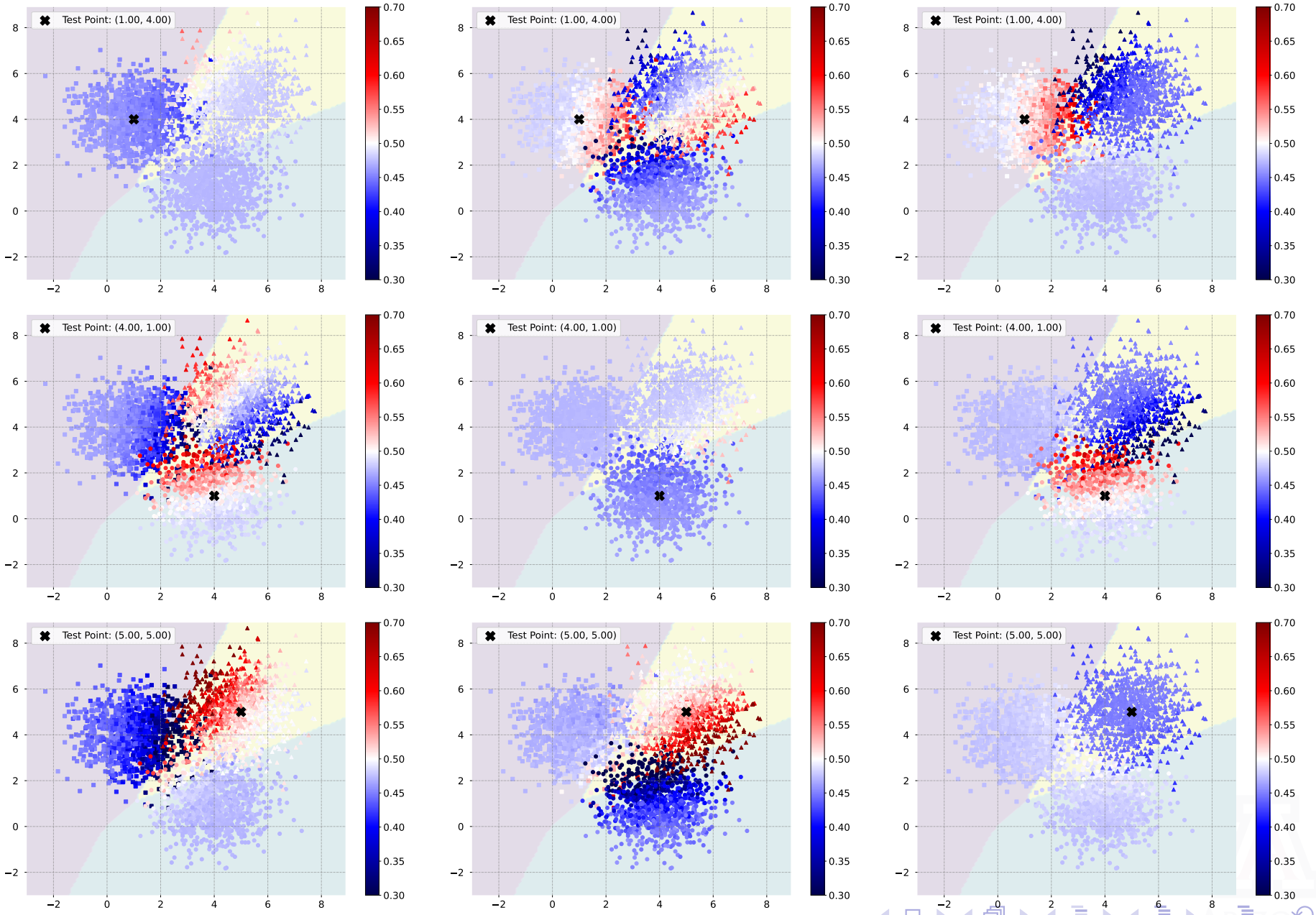


Figure: Updated predictions with kernel a_i updated via gradient descent with training data overlaid for classes 1 (left), 2 (middle), and 3 (right). The high prediction confidence in regions far from training points demonstrates that the learned kernel is non-stationary.

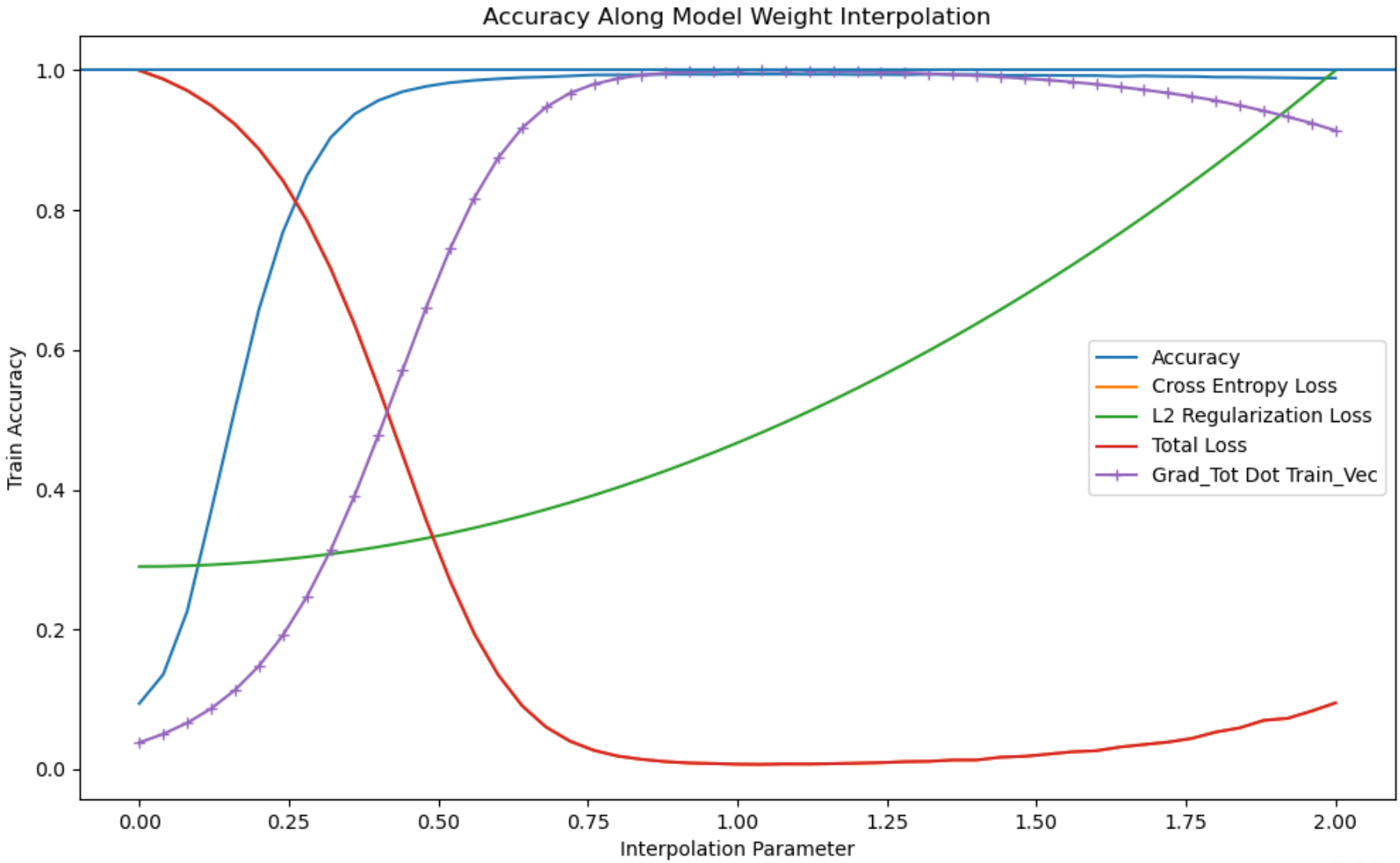
Kernel Properties



Experimental Results



Gradient Alignment During Training



Contents

Contribitions

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 **??** 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.

Generalized Exact Path Kernel (gEPK)

Theorem (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:

$$f(x;\theta_S) = f(x;\theta_0) + \sum_{i=1}^{N}\sum_{s=1}^{S} \varepsilon \left(\int_0^1 \varphi_{s,t}(x) dt \right) \frac{dL(f(x_i,\theta_s), y_i)}{d\hat{y}_{\theta_s(0)}} (\varphi_{s,0}(x_i)) \tag{45}$$

$$\varphi_{s,t}(x) \equiv \nabla_\theta f(x;\theta_s(t)), \tag{46}$$

$$\theta_s(t) \equiv \theta_s(0) + t(\theta_{s+1}(0) - \theta_s(0)), \text{ and} \tag{47}$$

$$\hat{y}_{\theta_s(0)} \equiv f(x;\theta_s(0)). \tag{48}$$

OOD with Parameter Gradient Subspace

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

$$\left\{ \frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i); i \in \{1, ..., N\}, s \in \{1, ..., S\} \right\} \tag{49}$$

spans the subspace in which all model predictions can vary.

Examining Cutting Edge OOD with gEPK

GradNorm : computes the gradient of Kullback–Leibler divergence with respect to model parameters:

$$\frac{1}{C}\sum_{i}^{C}\frac{\partial L_{CE}(f(x;\theta),i)}{\partial \hat{y}}\nabla_{\theta}f(x;\theta) \tag{50}$$

This represents an average across a truncation of the left side of the inner product from the gEPK, later methods perform better using a more complete basis.

Examining Cutting Edge OOD with gEPK

ReAct, DICE, ASH, and VRA : Along with other recent work (Sun et al., 2021; Sun and Li, 2022; Xu et al., 2023a), some of the cutting edge for OOD as of early 2023 involves activation truncation techniques. 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. 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.

Examining Cutting Edge OOD with gEPK

GradOrth: Behpour et al. (2023) explicitly create a reference basis from parameter gradients on training data for comparison:

$$\nabla_\theta L(x, y) = (\theta x - y)x^T = \Omega x^T \tag{51}$$

Treating Ω 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 $R^L_{ID} = [x_1, x_2, ..., x_n]$ 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

$$\left\| U^L_{ID}\Sigma^L_{ID,k}(V^L_{ID})^T \right\|^2_F \geq \epsilon_{\text{th}} \|R^L_I D\|^2_F. \tag{52}$$

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

$$O(x) = (\nabla_{\theta_L} \mathcal{L}(f(x, \theta_L), y))S^L(S^L)^T \tag{53}$$

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:

$$\text{span}\left(\left\{\frac{dL(\hat{y}_i, y_i)}{d\hat{y}_i}\varphi_{s,0}(x_i) : x_i \in X_T, 0 \leq s \leq S\right\}\right). \tag{54}$$

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.

OOD is enabled by Parameter Gradients

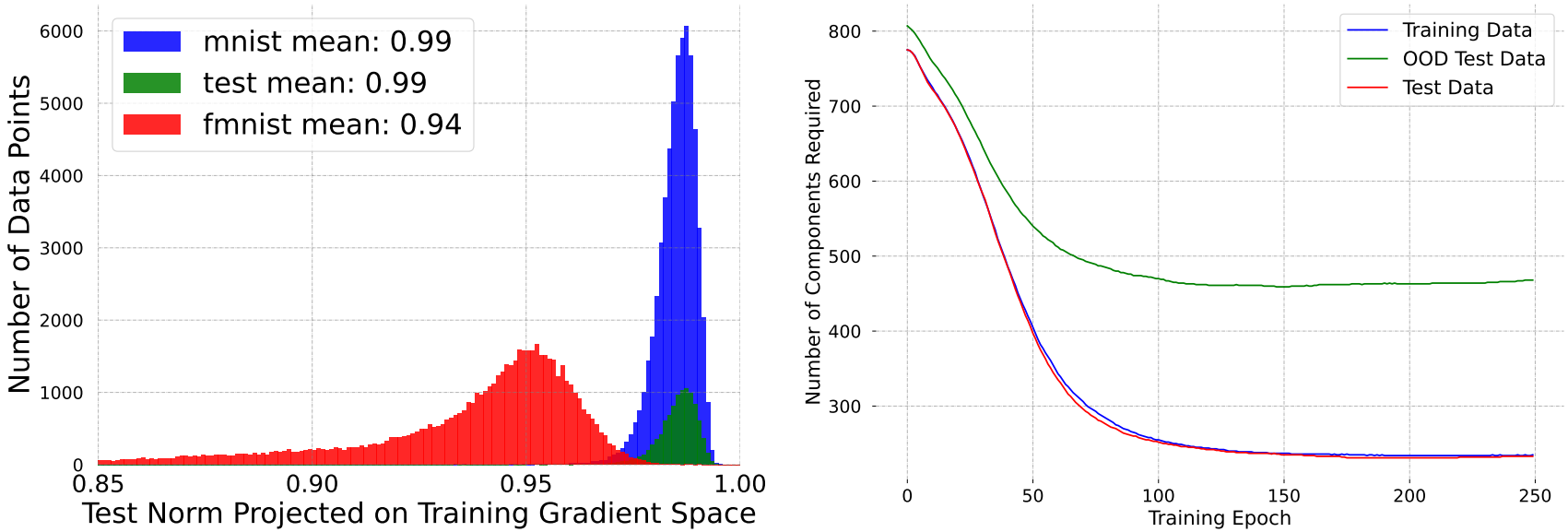


Figure: OOD detection using difference in training vs. test gradients. As the purpose of this paper is not to develop state of the art OOD detection methods, a comparison with recent benchmarks is not provided. Instead, a proof of concept that the gEPK can perform OOD detection is given. Left histogram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

Measuring Dimensionality with Parameter Gradient Space

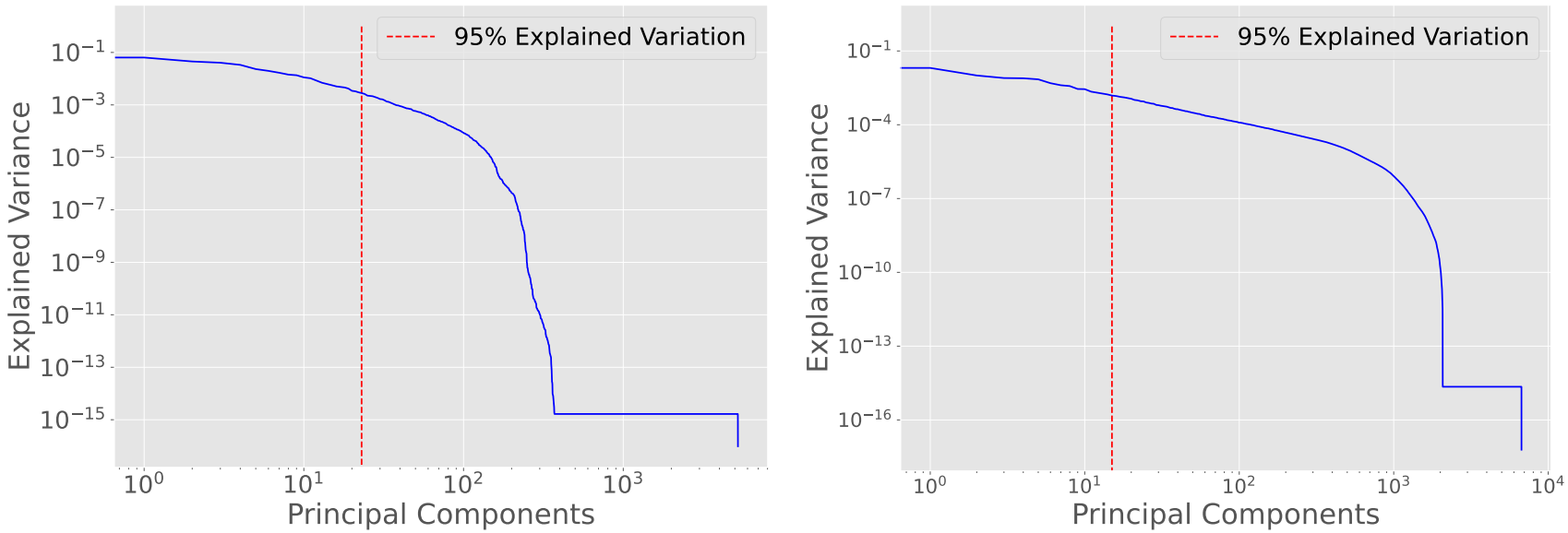


Figure: Explained Variance Ratio of parameter gradients. Left: MNIST, Right: CIFAR. 95% of variation can be explained with a relatively low number of components in both cases.

Computing Training Tangent (Dual) Space Basis

Given a test point x, we can evaluate its subspace by taking, for each x_i:

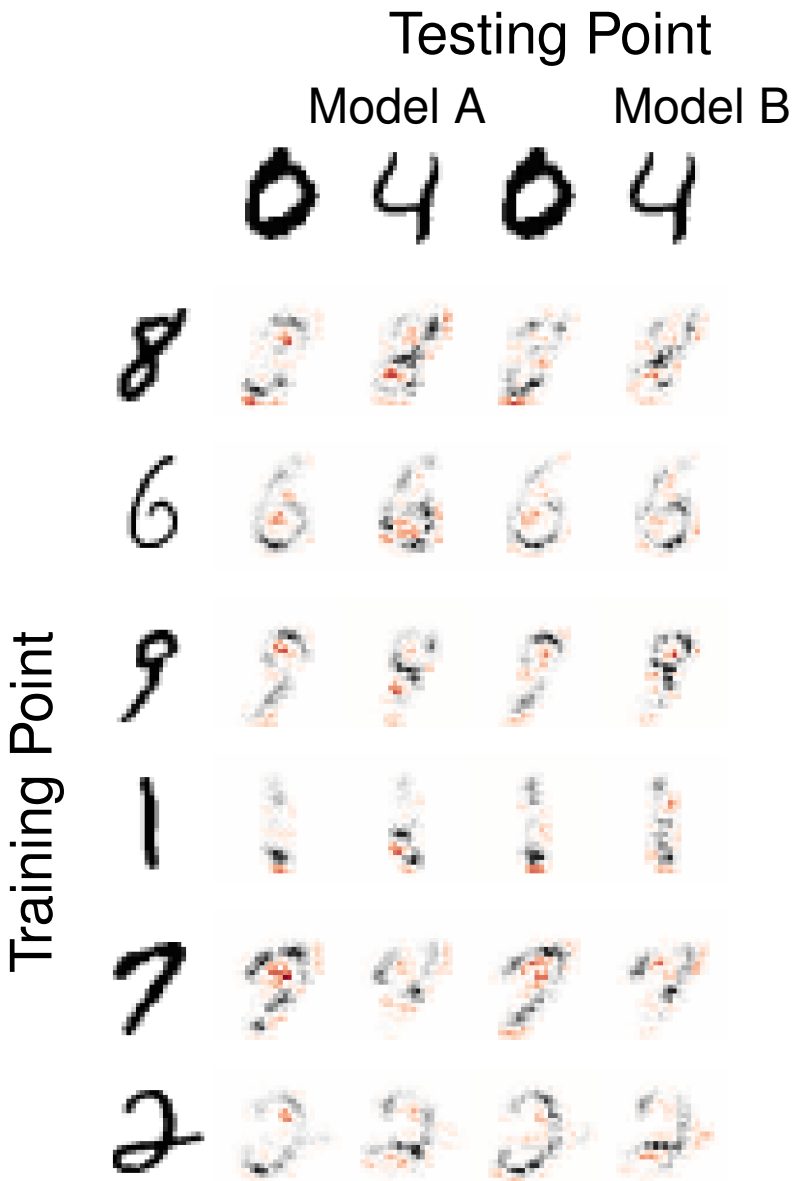
$$\frac{df(x,\theta_{\text{trained}})}{dx_j} = \frac{df(x,\theta_0(0))}{dx_j} + \sum_i \sum_s \int_0^1 \frac{d\left(\varphi_{s,t}(x)\frac{dL(x_i,y_i)}{df(x_i,\theta_s(0))}\varphi_{s,0}(x_i)\right)}{dx_j} d \tag{55}$$

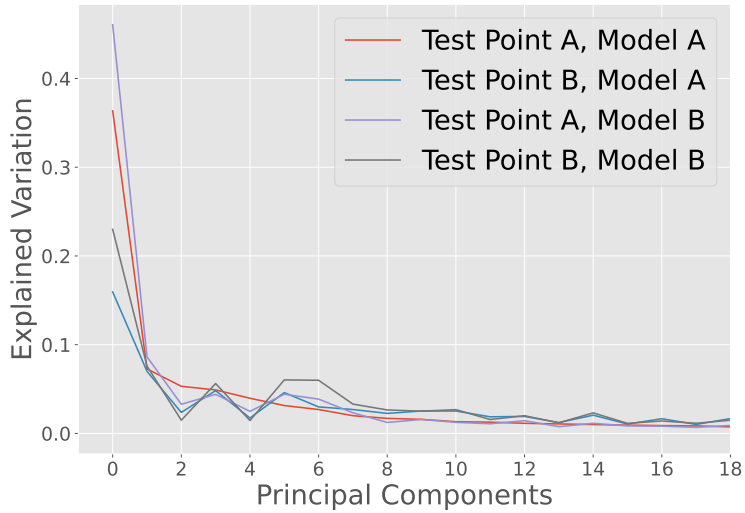
$$= \sum_i \sum_s \int_0^1 \varphi_{s,t}(x)dt \left(\frac{d^2L(x_i,y_i)}{df(x_i,\theta_s(0))dx_j}\varphi_{s,0}(x_i) + \frac{dL(x_i,y_i)}{df(x_i,\theta_s(0))} \tag{56}$$

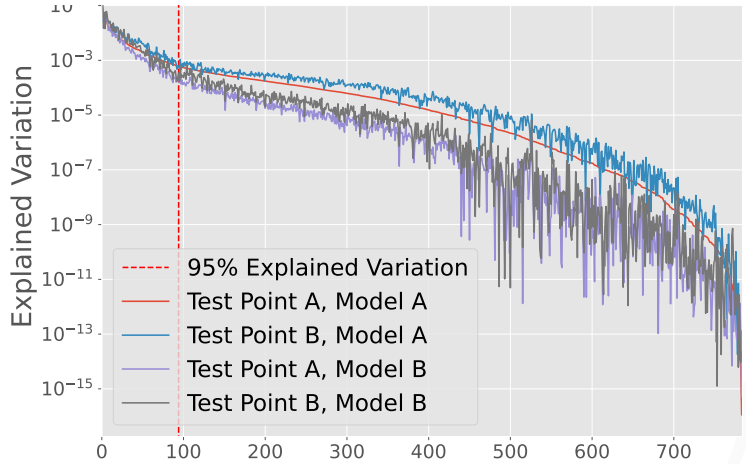
We can see that these gradients will be zero except when $i = j$, thus we may summarize these gradients as a matrix (tensor in the multi-class case), G, with

$$G_j = \sum_s \int_0^1 \varphi_{s,t}(x)dt \left(\frac{d^2L(x_i,y_i)}{df(x_i,\theta_s(0))dx_j}\varphi_{s,0}(x_i) + \frac{dL(x_i,y_i)}{df(x_i,\theta_s(0))}\frac{d\varphi_{s,0}(x_i)}{dx_j}\right) \tag{57}$$

Measuring Dimensionality







Measuring Dimensionality

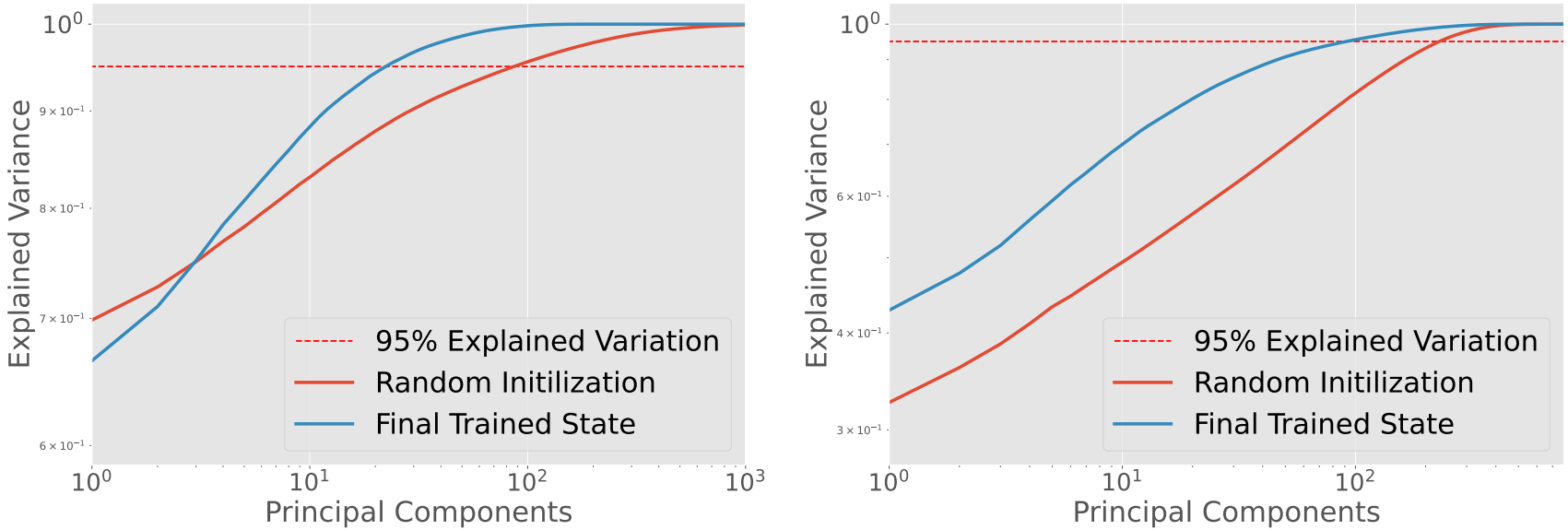
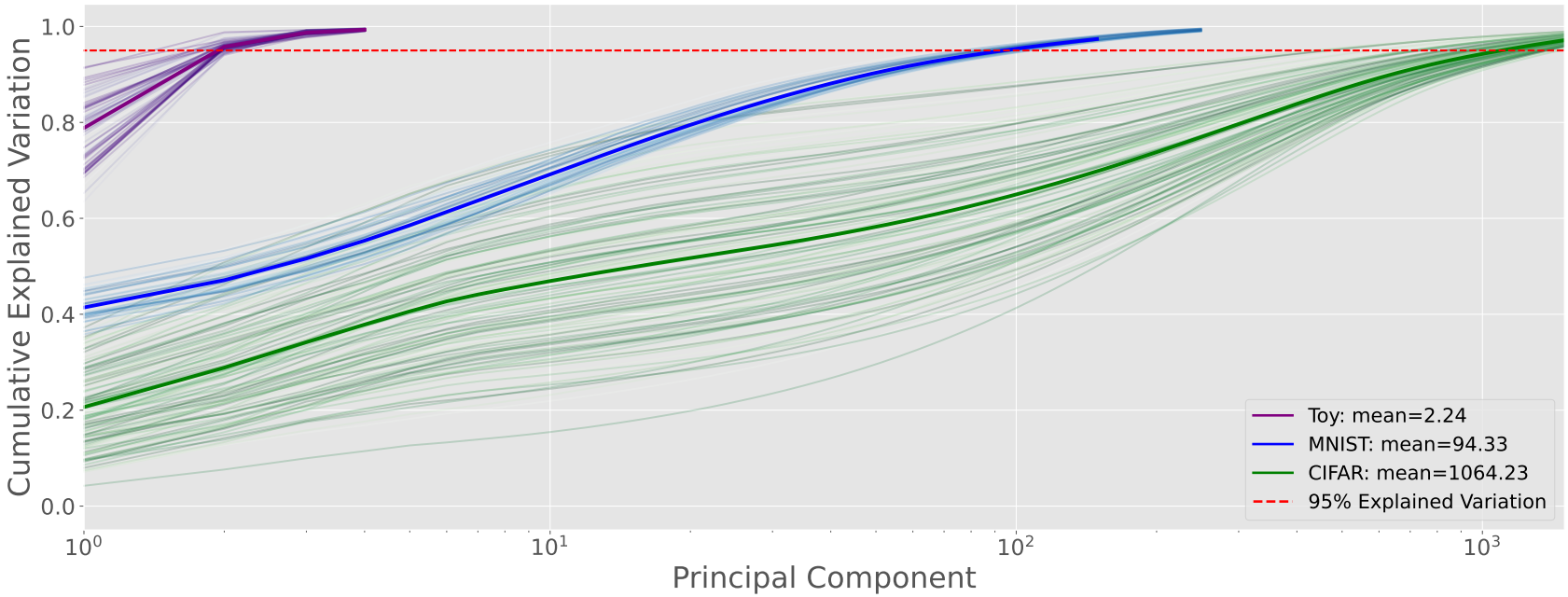


Figure: Differences between observing gradients in input space vs. weight space. Left: CDF of explained variation parameter space. Right: CDF of explained variation input space. Red solid line indicates a model at random initialization while the blue solid line represents the fully trained state. From 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 et al., 2021).

Measuring Dimensionality with Training Tangent Dual Space



Reducing the gEPK with SVD

Take a model, $f(x, \theta)$, which satisfies the necessary conditions for expression as:

$$f(x, \theta_{\text{tr}}) = f(x, \theta_0(0)) + \sum_s \int_0^1 \left\langle \varphi_{s,t}(x), \sum_i \left(\frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))} \varphi_{s,0}(x_i) \right) \right\rangle dt \tag{58}$$

$$\varphi_{s,t}(x) = \nabla_\theta f(x, \theta_s(t)) \tag{59}$$

Then we will let the RHS be rows of a matrix $L \cdot A$ where $L = \text{diag}(\frac{dL(x_i, y_i)}{df(x_i, \theta_s(0))})$ and each row $A_i = \varphi_{s,0}(x_i)$. Then we can compute $U \Sigma V^T = A$. We will truncate V and rewrite

$$f(x, \theta_{\text{tr}}) = f(x, \theta_0(0)) + \sum_s \int_0^1 \left\langle \varphi_{s,t}(x), \sum_k a_k \sigma_k V_k \right\rangle dt \tag{60}$$

(61)

Where a_k is computed from the loss gradients and σ_k is the k^{th} singular value.

Contents

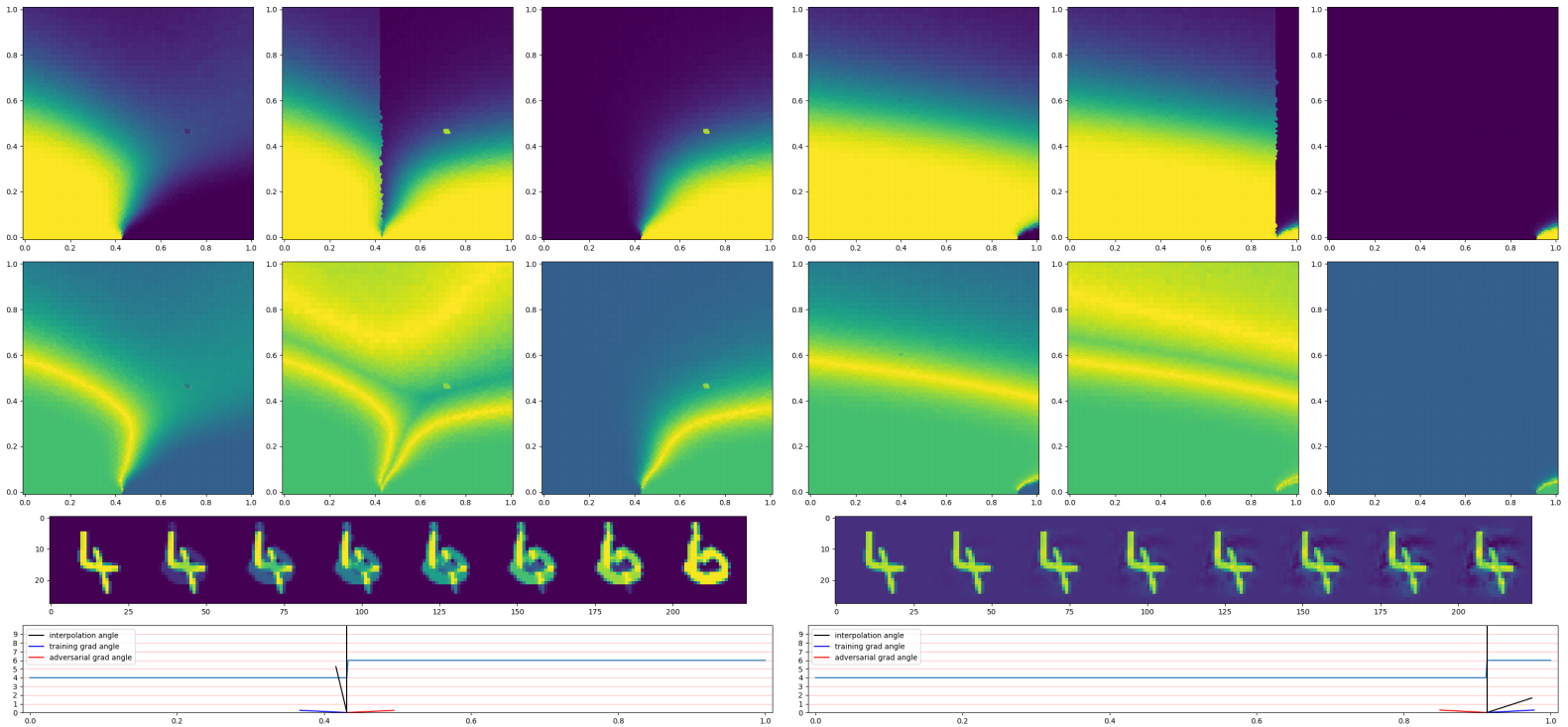
Conclusion : Next Direct Results

- In addition to making the above replacement with SVD, random features may be a cheap alternative with similar performance.
- A more general spectral form is possible, using features which span multiple training steps. Random Fourier Features may be a nice cheap way to accomplish this.
- Decomposition of (adversarial) input gradients according to the principal components from the training inputs (or principal components using the SVD formulation above).

$$f(x,\theta_F) = f(x,\theta_0) + \sum_i \sum_s \int_0^1 \langle \nabla_\theta f(x,\theta_s(t)), f(x_i,\theta_s(t)) \rangle dt \quad (62)$$

$$\frac{df(x,\theta_F)}{dx} = \frac{df(x,\theta_0)}{dx} + \sum_i \sum_s \int_0^1 \langle \nabla_\theta \frac{df(x,\theta_s(t))}{dx}, f(x_i,\theta_s(t)) \rangle dt. \quad (63)$$

Conclusion : Next Applications



Decision boundary curvature can now be analyzed directly using decompositions based on the gEPK.

Conclusion

Today, you have heard about

- Geometric observations about adversarial attacks using a novel soft spatial metric.
- A brand new method for representing neural networks in terms of blinear maps which admits useful decompositions and provides a robust theoretical framework for analysis.
- Analysis explaining cutting edge OOD detection using this kernel representation framework and showing that these techniques are in general subsets or reductions of the subspaces revealed by this representation.
- Connections between geometric properties and robustness which can be explored by applying these techniques.

Further Future Work

- Due to the asymmetry observed in this kernel method, these functionals cannot be limited to Hilbert space.
- Derive a Banach space with reproducing kernels in which these functionals can be embedded.
- Leverage the properties of this banach space to apply convergence and accuracy bounds.

Citations I

Attali, J.-G. and Pagès, G. (1997). Approximations of functions by a multilayer perceptron: a new approach. *Neural networks*, 10(6):1069–1081.

Behpour, S., Doan, T., Li, X., He, W., Gou, L., and Ren, L. (2023). Gradorth: A simple yet efficient out-of-distribution detection with orthogonal projection of gradients. *CoRR*, abs/2308.00310.

Bengio, Y. et al. (2009). Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127.

Blum, A., Dick, T., Manoj, N., and Zhang, H. (2020). Random smoothing might be unable to certify l^∞ robustness for high-dimensional images. *The Journal of Machine Learning Research*, 21(1):8726–8746.

Chen, Y., Huang, W., Nguyen, L., and Weng, T.-W. (2021). On the equivalence between neural network and support vector machine. *Advances in Neural Information Processing Systems*, 34:23478–23490.

Chizat, L. and Bach, F. (2020). Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In Abernethy, J. and Agarwal, S., editors, *Proceedings of Thirty Third Conference on Learning Theory*, volume 125 of *Proceedings of Machine Learning Research*, pages 1305–1338. PMLR.

Citations II

Cohen, J., Rosenfeld, E., and Kolter, Z. (2019). Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR.

Collobert, R., Bengio, S., and Mariéthoz, J. (2002). Torch: a modular machine learning software library.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. IEEE.

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

Domingos, P. (2020b). Every model learned by gradient descent is approximately a kernel machine. *CoRR*, abs/2012.00152.

Ganz, R., Kawar, B., and Elad, M. (2022). Do perceptually aligned gradients imply adversarial robustness? *arXiv preprint arXiv:2207.11378*.

Citations III

Ghojogh, B., Ghodsi, A., Karray, F., and Crowley, M. (2021). Reproducing kernel hilbert space, mercer's theorem, eigenfunctions, nyström method, and use of kernels in machine learning: Tutorial and survey.

Gilmer, J., Metz, L., Faghri, F., Schoenholz, S. S., Raghu, M., Wattenberg, M., and Goodfellow, I. (2018). Adversarial spheres. *arXiv preprint arXiv:1801.02774*.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*. arXiv: 1412.6572.

He, W., Li, B., and Song, D. (2018). Decision boundary analysis of adversarial examples. In *International Conference on Learning Representations*.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

Citations IV

Hu, S., Yu, T., Guo, C., Chao, W., and Weinberger, K. Q. (2019). A new defense against adversarial images: Turning a weakness into a strength. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32 (NeurIPS 2019) Vancouver, BC, Canada*, pages 1633–1644.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. (2019). Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32:125–136.

Incudini, M., Grossi, M., Mandarino, A., Vallecorsa, S., Pierro, A. D., and Windridge, D. (2022). The quantum path kernel: a generalized quantum neural tangent kernel for deep quantum machine learning.

Ivakhnenko, A. G. and Lapa, V. G. (1965). *Cybernetic predicting devices*. CCM Information Corporation.

Jacot, A., Gabriel, F., and Hongler, C. (2018). Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31.

Citations V

Kaur, S., Cohen, J., and Lipton, Z. C. (2019). Are perceptually-aligned gradients a general property of robust classifiers? *arXiv preprint arXiv:1910.08640*.

Khoury, M. and Hadfield-Menell, D. (2018). On the geometry of adversarial examples. *CoRR*, abs/1811.00525.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25.

Kumar, A., Levine, A., Goldstein, T., and Feizi, S. (2020). Curse of dimensionality on randomized smoothing for certifiable robustness. In *International Conference on Machine Learning*, pages 5458–5467. PMLR.

Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*. arXiv: 1607.02533.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

Citations VI

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D., and Jana, S. (2019). Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE.

Li, B., Chen, C., Wang, W., and Carin, L. (2019). Certified adversarial robustness with additive noise. *Advances in neural information processing systems*, 32.

Linnainmaa, S. (1970). The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7.

McClelland, J. L., Rumelhart, D. E., Group, P. R., et al. (1986). Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271.

Citations VII

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

Minsky, M. and Papert, S. (1969). Perceptrons: AnIntroduction to computational geometry. *MITPress, Cambridge, Massachusetts*.

Nair, V. and Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. pages 807–814.

Petersen, P. and Voigtlaender, F. (2018). Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330.

Rasmussen, C. E., Williams, C. K., et al. (2006). *Gaussian processes for machine learning*, volume 1. Springer.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Roth, K., Kilcher, Y., and Hofmann, T. (2019). The odds are odd: A statistical test for detecting adversarial examples. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97, pages 5498–5507.

Citations VIII

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117.

Shafahi, A., Huang, W. R., Studer, C., Feizi, S., and Goldstein, T. (2018). Are adversarial examples inevitable? *CoRR*, abs/1809.02104.

Shah, H., Jain, P., and Netrapalli, P. (2021). Do input gradients highlight discriminative features? *Advances in Neural Information Processing Systems*, 34:2046–2059.

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

Shamir, A., Melamed, O., and BenShmuel, O. (2021). The dimpled manifold model of adversarial examples in machine learning. *CoRR*, abs/2106.10151.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Citations IX

Sun, Y., Guo, C., and Li, Y. (2021). React: Out-of-distribution detection with rectified activations. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 144–157.

Sun, Y. and Li, Y. (2022). DICE: leveraging sparsification for out-of-distribution detection. In Avidan, S., Brostow, G. J., Cissé, M., Farinella, G. M., and Hassner, T., editors, *Computer Vision - ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23-27, 2022, Proceedings, Part XXIV*, volume 13684 of *Lecture Notes in Computer Science*, pages 691–708. Springer.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

Citations X

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2014). Intriguing properties of neural networks. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Tjeng, V., Xiao, K., and Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*.

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., and Madry, A. (2018). Robustness may be at odds with accuracy. *stat*, 1050:11.

Wang, Y., Zou, D., Yi, J., Bailey, J., Ma, X., and Gu, Q. (2020). Improving adversarial robustness requires revisiting misclassified examples. In *International Conference on Learning Representations*.

Werbos, P. J. (1974). Beyond regression: New tools for prediction and analysis in the be havioral sciences/'phd diss., harvard uni versity. werbos, paul j. 1988. *Generalization of back propagation with application to a recurrent gas market method," Neural Networks*, 1(4):339–356.

Citations XI

Xu, M., Lian, Z., Liu, B., and Tao, J. (2023a). Vra: Variational rectified activation for out-of-distribution detection.

Xu, Y., Sun, Y., Goldblum, M., Goldstein, T., and Huang, F. (2023b). Exploring and exploiting decision boundary dynamics for adversarial robustness.

Yang, G., Duan, T., Hu, J. E., Salman, H., Razenshteyn, I., and Li, J. (2020). Randomized smoothing of all shapes and sizes. In *International Conference on Machine Learning*, pages 10693–10705. PMLR.