Neighborhoods of Adversarial Examples

Brian Bell : University of Arizona

June 18, 2023

Contents

Part I
Persistence

1 Artificial Neural Networks ANNs

2 Adversarial Attacks

3 Defining adversarial attacks mathematically

4 Persistence metric for Adversarial Attacks

5 Persistence results

6 Introduction

Artificial Neural Networks and other optimization-based general function approximation models are the core of modern machine learning [Prakash et al.(2018)Prakash, Moran, Garber, DiLillo, and Storer]. These models have dominated competitions in image processing, optical character recognition, object detection, video classification, natural language processing, and many other fields [Schmidhuber(2015)]. All such modern models are trained via gradient-based optimization, e.g. Stochastic Gradient Descent (SGD) with gradients computed via back propagation. [Goodfellow et al.(2013)Goodfellow, Bulatov, Ibarz, Arnoud
Although the performance of these models is practically miraculous within the training and testing context for which they are designed, they have a few intriguing properties. It was discovered in 2013 [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus] that images can be generated which apparently trick such models in a classification context in difficult-to-control ways [Khoury and Hadfield-Menell(2018)]. The intent of this research is to investigate these *adversarial examples* in a mathematical context and use them to study pertinent properties of the learning models from which they arise.

7 Artificial Neural Networks (ANNs)

The history of Neural Networks begins in the field of Theoretical Neuropsychology with a much-cited paper by McCulloch and Pitts in which they describe the mechanics of cognition in the context of computation [McCulloch and Pitts(1943)]. This initial framework for computational cognition did not include a notion for learning, but the following decade brought the concept of learning (as optimization) and many simple NNs (linear regression models applied to computational cognition). The perceptron, the most granular element of a neural network, was proposed in another much-cited paper by Rosenblatt in 1958 [Rosenblatt(1958)], and multilevel (deep) networks were proposed by 1965 in a paper by Ivakhnenko and Lapa [Ivakhnenko and Lapa(1965)].

By the 1960s, these neural network models became disassociated from the cutting-edge of cognitive science, and interest had shifted to their application in modeling and industrial computation. The hardware limitations of the time served as a significant barrier to wider application and the concept of the "neural network" was sometimes treated as a solution looking for a problem. Compounding these limitations was a significant roadblock published by Minsky and Papert in 1969: A proof that basic perceptrons could not encode exclusive-or [Minsky and Papert(1969)].

As a result, interest in developing neural network theory waned. The next necessary step in the development of modern neural network models was an advance that would allow them to be trained efficiently with computing power available. Learning methods required a gradient, and the technique necessary for computing gradients of large-scale multi-parameter models was apparently proposed in a 1970 in a Finnish masters thesis [Linnainmaa(1970)]. Techniques from control theory were applied to develop a means of propagating error backward through models which could be described as directed graphs. The idea was applied to neural networks by a Harvard student Paul Werbos[Werbos(1974)] and refined in later publications.

The final essential puzzle piece for neural network models was to take advantage of their layered structure, which would allow backpropagation computations at a given layer to be done in parallel. This key insight, indeed the core of much of modern computing, was a description of parallel and distributed processing in the context of cognition by Rumelhard and McClelland in 1986 [McClelland et al.(1986)McClelland, Rumelhart, Group, et al.] with an astonishing 22,453 citations (a number that grows nearly every day). With these pieces in place, the world was ready for someone to finally apply neural network models to a relevant problem. In 1989, Yann LeCun and a group at Bell Labs managed to do just that. LeCun refined backpropagation into the form it is used today [LeCun et al.(1989)LeCun, Boser, Denker, Henderson, Howard, Hubbard, and Jackel], invented Convolutional Neural Networks 7.2 [LeCun et al.(1995)LeCun, Bengio, et al.], and by 1998, he had worked with his team to implement what has become the industry standard for banks to recognize hand-written numbers on checks [LeCun et al.(1998)LeCun, Bottou, Bengio, Haffner, et al.].

Starting in the 2000s, neural networks have blown up in scale and application, so it's harder to keep track of the discrete historical developments. Most of the progress in terms of performance and application has come from contests (e.g: `http://image-net.org/challenges/LSVRC/`) in which a bounty (or publicity) is offered and labs around the world compete to build a network which solves the competition's problem. Schimdhuber's group [Schmidhuber(2015)], and a similar group at Google have dominated many of these competitions. The cutting-edge today is represented by networks like Inception v4 designed by Google for image classification which contains approximately 43 million parameters [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus]. Early versions of this network took 1-2 million dollars worth of compute-time to train. ANNs now appear in nearly every industry from devices which use ANNs to intelligently adapt their performance, to the sciences which rely on ANNs to eliminate tedious sorting and identification of data that previously had to be relegated to humans.

7.1 Structure

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

A *neuron* is a nonlinear operator that takes input in $\mathbb{R}^n$ to $\mathbb{R}$, historically designed to emulate the activation characteristics of an organic neuron. A collection of neurons that are connected via a (usually directed) graph structure are known as an *Artificial Neural Network (ANN)*.

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

Definition 7.1. *A **Perceptron** is a function $P_{\vec{w}} : \mathbb{R}^n \to \mathbb{R}$ which has* weights *$\vec{w} \in \mathbb{R}^n$ corresponding with each element of an input vector $\vec{x} \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$:*

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

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

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

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

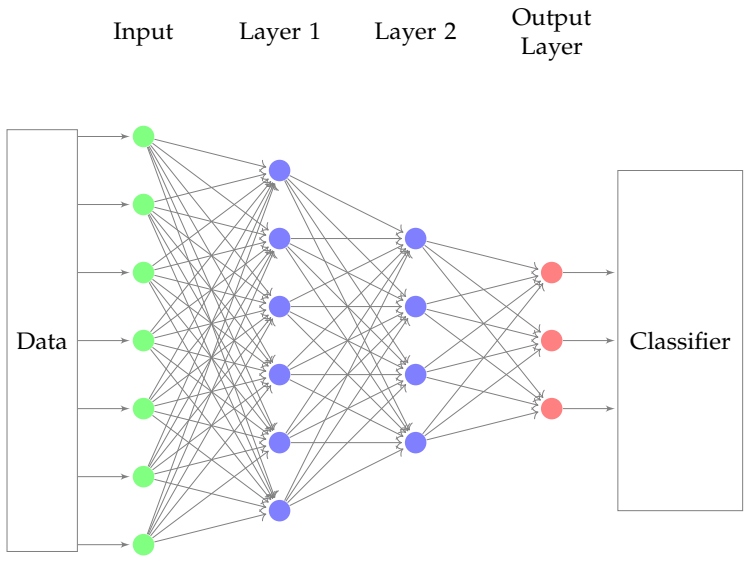
Definition 7.2. *The Rectified Linear Unit (ReLU) function is*

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

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

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

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



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

Definition 7.3. *Softmax (or the normalized exponential) is the function given by*

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

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

Definition 7.4. *We can define a classifier which picks the class corresponding with the largest output element from Softmax:*

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

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

[Bishop(2006)]

7.2 Convolutional Neural Networks (CNNs)

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

8 Training ANNs

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

A breakthrough came with the application of techniques derived from control theory to ANNs in the late 1980s [Rumelhart et al.(1986)Rumelhart, Hinton, and Williams], dubbed back-propagation. This technique was refined into its modern form in the thesis and continuing work of Yann LeCun [LeCun et al.(1988)LeCun, Touresky, Hinton, and Sejnowski]. In this method, error is propagated backward taking advantage of the directed structure of the network to compute a gradient for each parameter defining it. Because modern ANNs are usually separated into discrete layers, gradients can be computed in parallel for all perceptrons at the same depth of the network [Bishop(2006)]. Leveraging modern GPUs and parallel computing technologies, these gradients can be computed very quickly. There are a number of important considerations in training. We discuss a few in the following sections.

8.1 Selection of the Training Set

The first step in training an ANN is the selection of a training set. ANNs fundamentally are universal function approximators: Given a set of input data and corresponding output data, they approximate a mapping from one to the other. Performance is dependent on how well the phenomenon we hope to model is represented by the training data. The training data must consist of a set of inputs (e.g., images) and a set of outputs (e.g., labels) which contain sufficient examples

to characterize the intended model. In a way, this is how we pose a question to the neural network. One must always ask whether the question we wish to pose is well-expressed by the training data we have available.

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

8.2 Selecting a Loss Function

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

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

Definition 8.1. *The Cross-Entropy Loss comparing two possible outputs is* $L(y, \hat{y}) = -\sum_i y_i \log \hat{y}_i$.

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

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

Definition 8.2. *Given a loss function* L*, the Empirical Risk over a training dataset* (X, Y) *of size* N *is*

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

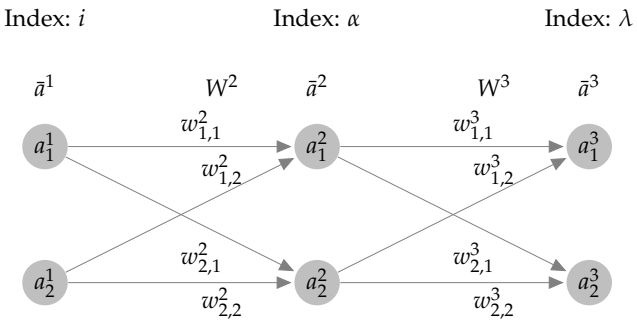
We seek parameters $\vec{w}$ which will minimize $R_{\text{emp}}(P_w(x))$. This will be done with gradient-based optimization.

8.3 Computation of Gradient via Backpropagation

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

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

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



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

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

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

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

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

$$\bar{a}^n = \bar{A}^n(W^n(\bar{A}^{n-1}(W^{n-1}(\cdots(\bar{A}^2(W^2\bar{a}^1))\cdots)))) \tag{3}$$

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

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

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

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

In matrix form, we have

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

Where $\odot$ signifies element-wise multiplication.

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

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

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.

8.4 Optimization of Weights

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

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

In this case, the step size (learning rate) η is fixed throughout training. Numerical computation of the gradient requires first evaluating the network forward by computing the output for a given input. The value of every node in the network is saved and these values are used to weight the error as it is propagated backward through the network. Once the gradient is computed, the weights are adjusted according to the step defined above. This process is repeated until convergence is attained to within a tolerance. It should be clear from the number of terms in this calculation that the initial guess and step size can have significant effect on the eventual trained weights. Due to lack of a guarantee for general convexity, poor guesses for such a large number of parameters can lead to gradients blowing up or down [Bishop(2006)]. . Due to nonlinearity and the plenitude of local minima in the loss function, classic gradient descent usually does not perform well during ANN training.
By far the most common technique for training the weights of neural networks adds noise in the form of random re-orderings of the training data to the general optimization process and is known as stochastic gradient descent.

Definition 8.3. *Stochastic Gradient Descent (SGD)*

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

Batch Stochastic Gradient Descent

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

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

9 Problem Setting: Adversarial Attacks

In general *Adversarial Attacks* against models are small perturbations from natural data which significantly perturb model output. Szegedy et al. [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goo realized that the same computational tools used to train ANNs could be used to generate attacks that would confuse them. Their approach was to define a loss function relating the output of the ANN for a given initial image to a target adversarial output plus the L^2-norm of the input and use backpropagation to compute gradients – not on the weights of the neural network, but on just the input layer to the network. The solution to this optimization problem, efficiently approximated by a gradient-based optimizer, would be a slightly perturbed natural input with a highly perturbed output. Their experimental results are striking:

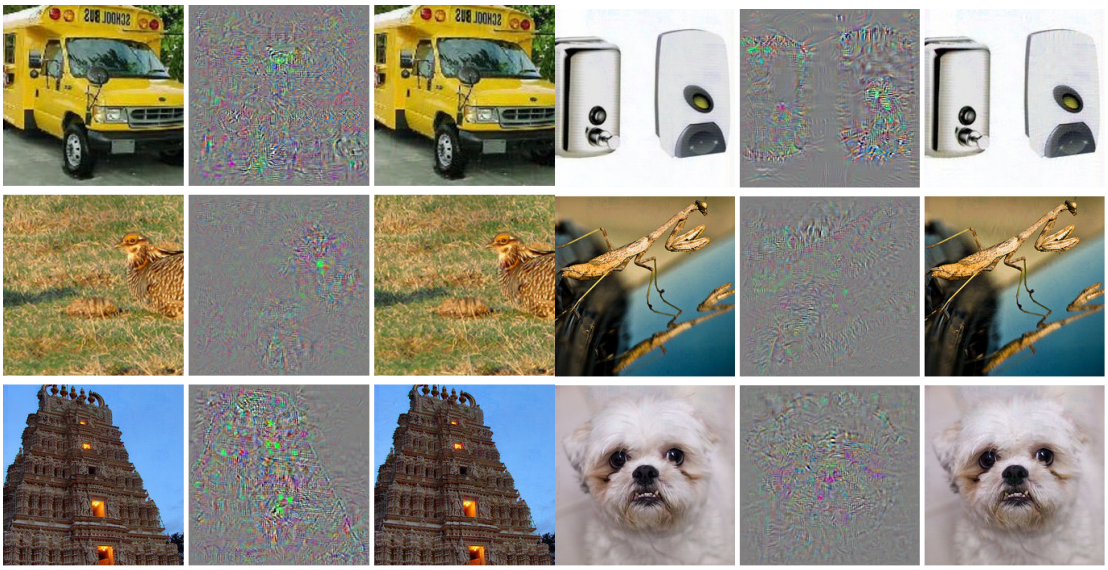


Figure 1: Natural Images are in columns 1 and 4, Adversarial images are in columns 3 and 6, and the difference between them (magnified by a factor of 10) is in columns 2 and 5. All images in columns 3 and 6 are classified by AlexNet as "Ostrich" [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus]

9.1 Data Sets

The Dataset used above is known as ImageNet – a large set of labeled images varying in size originally compiled for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC). This dataset and its many subsets has become a standard for image classification and feature identification experiments. In the experiments that follow, ImageNet will be featured alongside the Modified National Institute of Standards and Technology (MNIST) dataset which is a database of hand written digits often used to develop image processing and character recognition systems. This dataset is much lower resolution than ImageNet and is therefore experiments run much more quickly on it and require less complex input/output.

9.2 L-BFGS minimizing distortion

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

Let $f : \mathbb{R}^m \to \{1, ..., k\}$ be a classifier and assume f has an associated continuous loss function denoted by $\text{loss}_f : \mathbb{R}^m \times \{1, ..., k\} \to \mathbb{R}^+$ and l a target adversarial .
Minimize $\|r\|_2$ subject to:

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

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

9.2.1 L-BFGS: Mnist

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

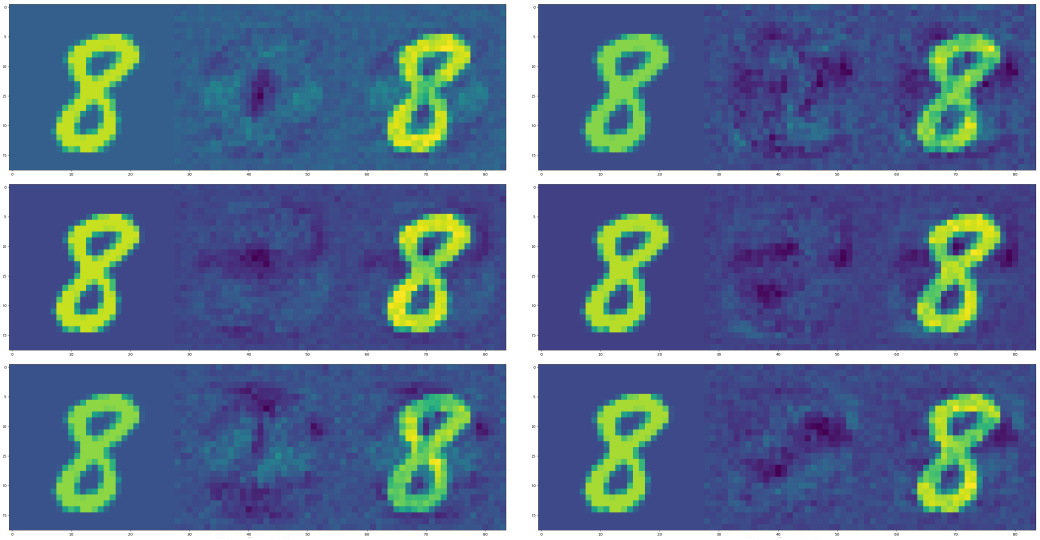


Figure 2: Original images on the left, Perturbation is in the middle, Adversarial Image (total of Original with Perturbation) is on the right. Column 1 shows an original 8 being perturbed to adversarial classes 0, 2, and 4. Column 2 shows adversarial classes 1, 3, and 5

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

Definition 9.1. *Distortion is the L^2 norm of the difference between an original image and a perturbed image, divided by the square root of the number of pixels in the image:*

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

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

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

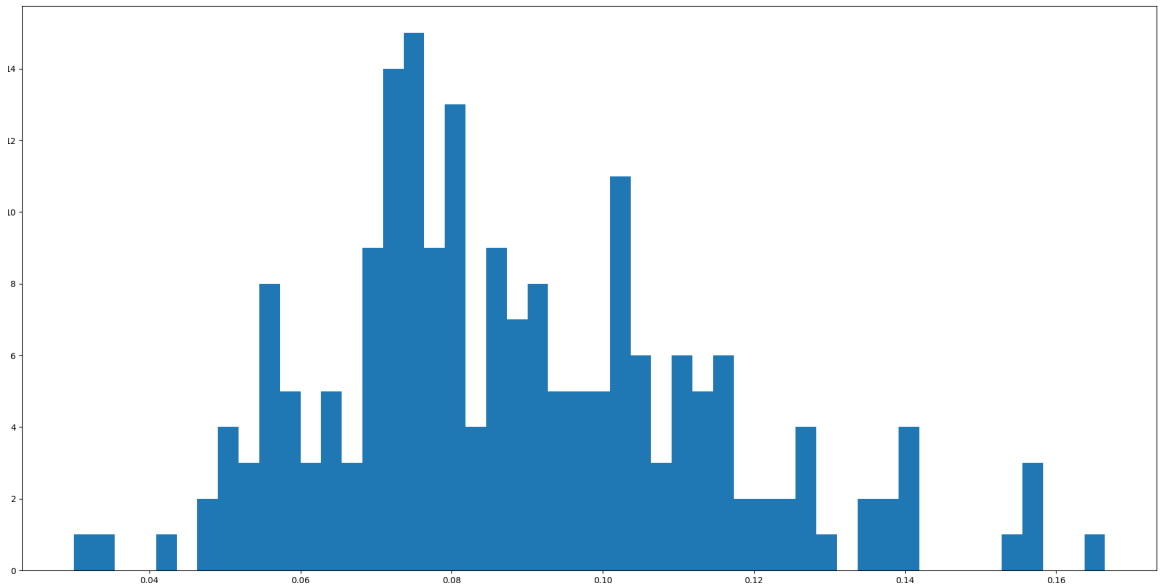


Figure 3: A histogram of the distortion measured for each of 900 adversarial examples generated using L-BFGS against the FC-200-200-10 network on Mnist. Mean distortion is 0.089.

9.2.2 L-BFGS: ImageNet

We also tried to replicate [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus results on ImageNet. Attacking VGG16, a well known model from the ILSVRC-2014 competition [Simonyan and Zisserman(2014)], on ImageNet images with the same technique generates the examples in figure 4:

Figure 4: Original images on the left, Perturbation (magnified by a factor of 100) by is in the middle, Adversarial Image (total of Original with Perturbation) is on the right.

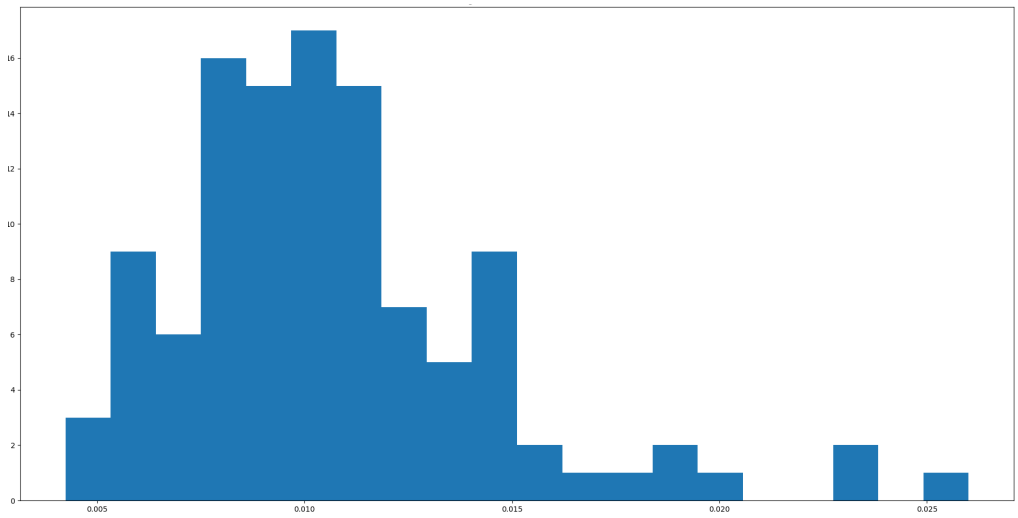


Figure 5: A histogram of the distortion measured for each of 112 adversarial examples generated using L-BFGS against the VGG16 network on ImageNet images with mean distortion 0.0107

9.2.3 Fast Gradient Sign Method (FGSM)

We also implemented a single step attack process uses the gradient of the loss function L with respect to the image to find the adversarial perturbation [Goodfellow et al.(2014)Goodfellow, Shlens, and Szegedy]. for given ε, the modified image $\hat{x}$ is computed as

$$\hat{x} = x + \epsilon \text{sign}(\nabla L(P_w(x), x)) \tag{4}$$

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

9.2.4 Iterative Gradient Sign Method (IGSM)

In [Kurakin et al.(2016)Kurakin, Goodfellow, and Bengio] 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:

$$x'_{m+1} = \text{Clip}_{x,\epsilon}\left\{x'_m + \alpha \times \text{sign}(\nabla \ell(F(x'_m), x'_m))\right\} \tag{5}$$

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

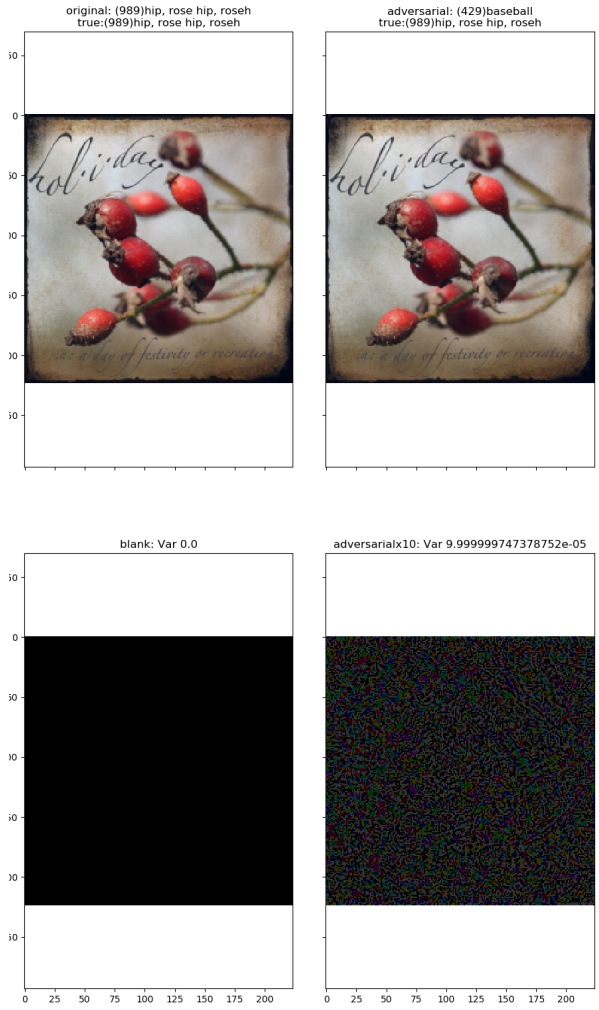


Figure 6: adversarial example generated against VGG16 (ImageNet) with IGSM. Original Image on the left, adversarial image and added noise (ratio of variance adversarial noise/original image: 0.0000999) on the right.

9.3 Other Attacks

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

9.3.1 Jacobian-based Saliency Map Attack (JSMA)

Another attack noted by [Papernot et al.(2015)Papernot, McDaniel, Jha, Fredrikson, Celik, and Swami] estimates the *saliency map*, a rating for each of the input features (e.g. each pixel) on how influential it is for causing the model to predict a particular class with respect to the model output [Wiyatno and Xu(2018)]. This attack modifies the pixels that are most salient. This is a targeted attack, and saliency is designed to find the pixel which increases the classifier's output for the target class while tending to decrease the output for other classes.

9.3.2 Deep Fool (DFool)

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

9.3.3 Carlini & Wagner (C&W)

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

$$||x - \hat{x}||_p + c * max\left(Z_k(\hat{x}_y) - max\{Z_k(\hat{x}) : k \neq y\}, -\kappa\right) \tag{6}$$

10 Exploring Stability under classifications

In this section we define and experimentally demonstrate a notion of stability of classification under a given classification model.

10.1 Definitions

To build up to a definition for stability, we develop definitions and terms to refer to adversarial examples without relying on subjective characteristics like human vision.Let X denote a space of possible data and C denote a set of classes,

Definition 10.1. *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 10.2. *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 $\mathcal{C}(\hat{x}) = c_t$.*

These definitions rely on a metric d which is commonly L^2, but can be other metrics. The following definition complements the definition for adversarial examples by providing a criteria for the local stability of adversarial examples:

Definition 10.3. *x is (γ, σ)-*stable *if $\mathbb{P}[\mathcal{C}(x') = \mathcal{C}(x)] \geq \gamma$ when $x' \sim \rho = N(X, \sigma^2 I)$ (i.e. x' is drawn from a Gaussian with variance σ and mean x).*

We see that

$$\mathbb{P}[\mathcal{C}(x') = \mathcal{C}(x)] = \int \chi_{\mathcal{C}^{-1}(\mathcal{C}(x))}(x') d\rho(x') = \rho(\mathcal{C}^{-1}\mathcal{C}(x))$$

In the case of images drawn from $\mathbb{R}^n$, we can write this integral precisely

$$\frac{1}{\sigma(2\pi)^{n/2}} \int_{\mathbb{R}^n} \chi_{\mathcal{C}^{-1}(\mathcal{C}(x))}(y) e^{\frac{|x-y|^2}{2\sigma^2}} d\vec{y}$$

This integral is approximated by taking N samples $x_k \sim \rho$ and computing

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

In our experiments, γ is fixed and σ is adjusted to determine the σ for which an image is $\gamma - \sigma$ stable. This is accomplished by a bracketing procedure, first expanding the search space bounds σ_{min} and σ_{max} so that the average number of samples at these bounds are below (respectively above) γ. This search space is then sequentially bisected to identify a σ corresponding with γ. The algorithm is presented in Appendix **??**

10.2 Stability Experiments

In the following experiments, adversarial examples are generated using IGSM (9.2.4) and L-BFGS methods (9.2.2). The neighborhoods around them are sampled using Gaussians with varying σ. Separately the bracketing algorithm described above is used to identify σs corresponding with $\gamma = 0.7$. The resultant σ statistic identifies examples as being $(\gamma = 0.7, \sigma)$ stable meaning that 70% of the samples drawn from a Gaussian with standard deviation σ are classified the same as the original class by the model.

10.2.1 FGSM Neighborhood sampling

In these initial experiments, the neighborhood around normal and adversarial examples are explored via sampling. An Mnist image x with classification 1 is selected, adversarial noise x_a is prepared using IGSM 5 for each target other than 1. The neighborhoods around each such image are examined by generating 1000 Gaussian samples at 1000 equally spaced σs.

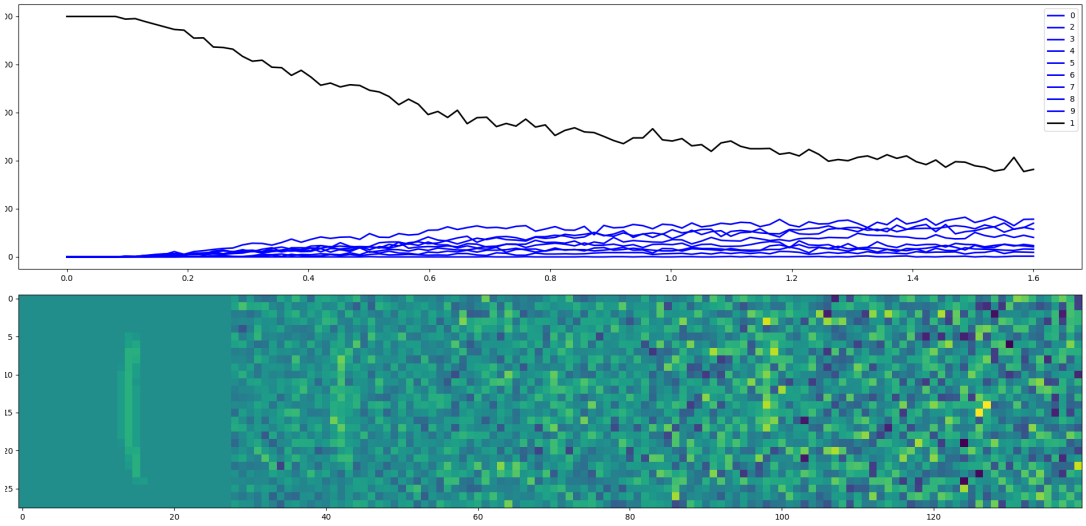


Figure 7: Plotting frequency of each class in samples with increasing variance around natural image. Original Image Class is shown as a black curve. Bottom shows example sample images.

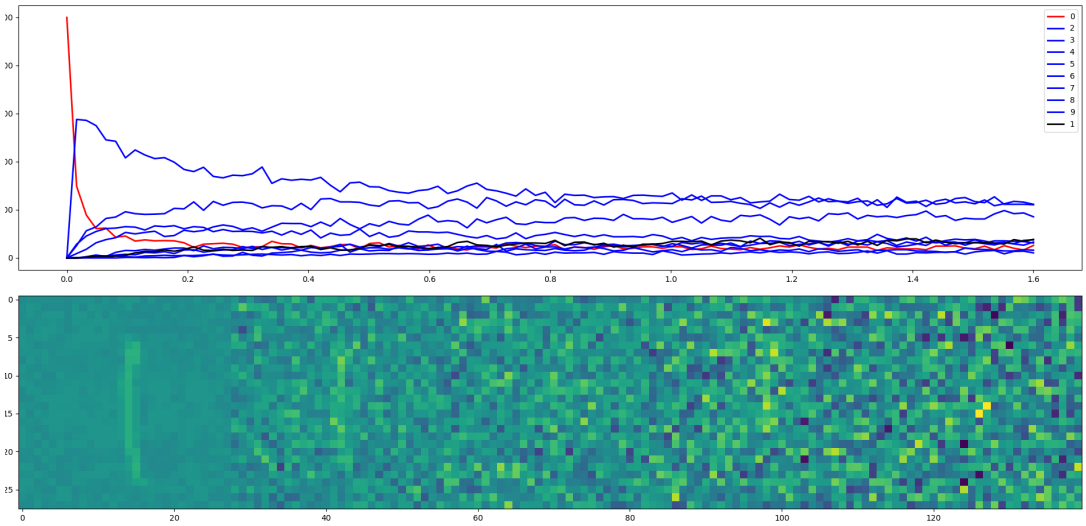


Figure 8: Plotting frequency of each class in samples with increasing variance around adversarial image. Adversarial class is shown as a red curve. Bottom shows example sample images.

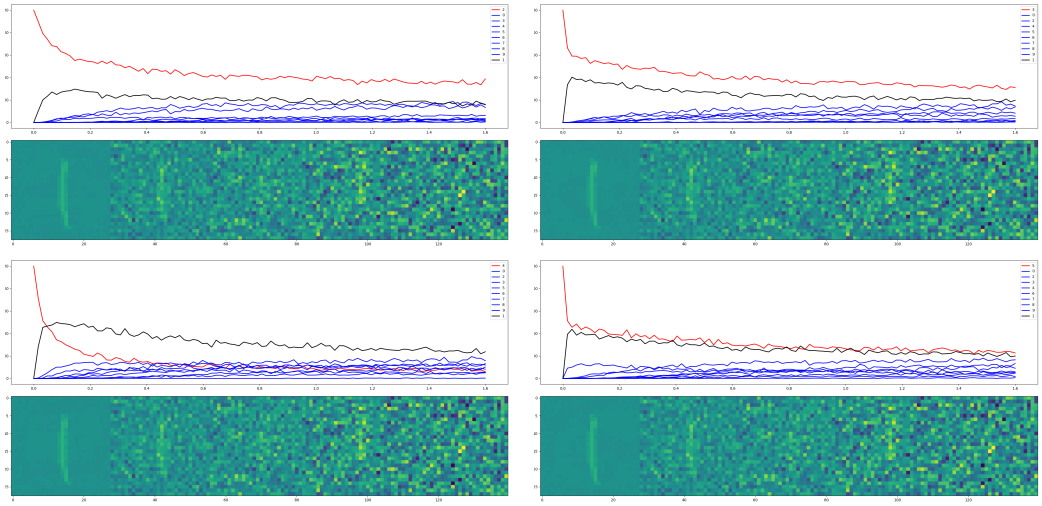


Figure 9: Plotting frequency of each class in samples with increasing variance around adversarial images. Original Class is shown as a black curve, adversarial in red, all others are blue. Bottom shows example sample images.

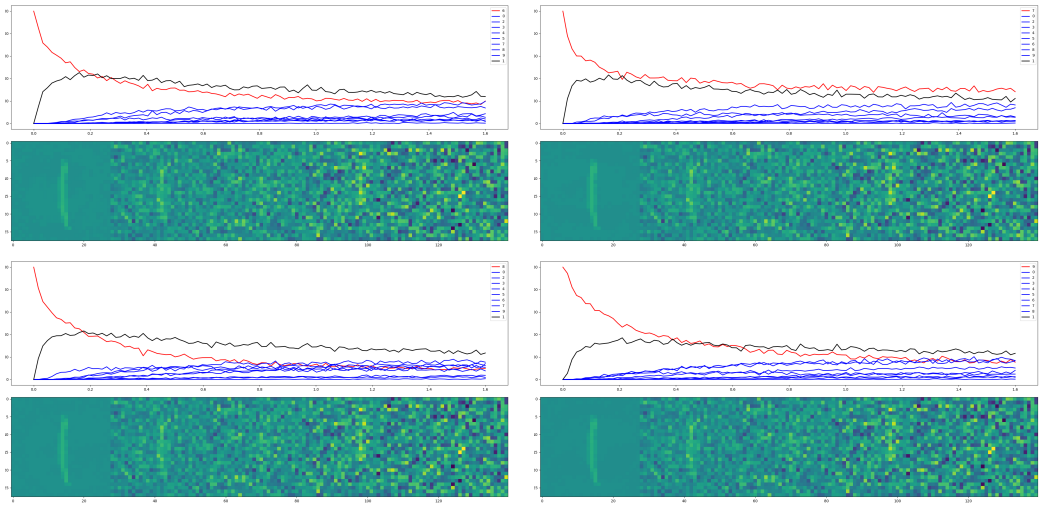


Figure 10: Plotting frequency of each class in samples with increasing variance around adversarial images. Original Class is shown as a black curve, adversarial in red, all others are blue. Bottom shows example sample images.

Attack examples are generated using IGSM (5) from each of 200 Mnist images to each possible target other than the original class (9 total targets) for a total of 1800 adversarial examples. 1800 natural images are selected with the same classes as the 1800 adversarial examples. The images are processed using the bracketing algorithm to identify σs corresponding with $\gamma = 0.7$ and the resulting sigmas are aggregated into the following histogram:

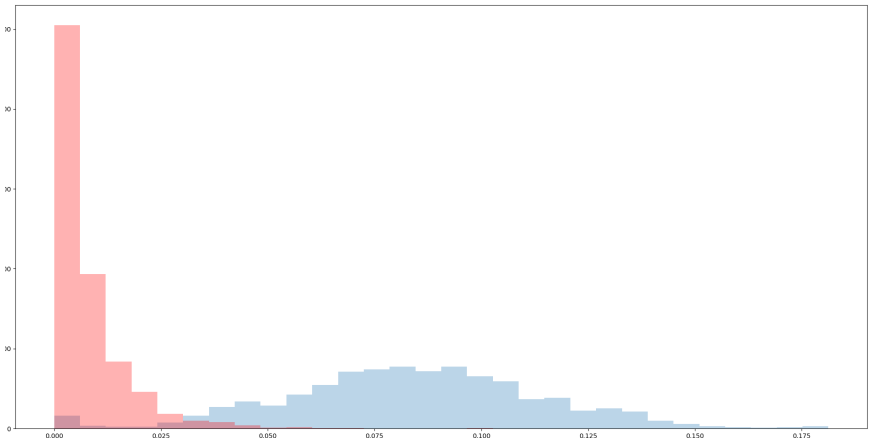


Figure 11: Histogram of σ for $(\gamma = 0.7, \sigma)$ stability of Adversarial Examples (Red) and Natural Examples (Blue)

In this experiment, the adversarial images (red in the histogram above) were much less stable than the natural images. In this way, 98% of the adversarial images could be identified by their $\gamma - \sigma$ stability characteristics.

10.2.2 Sampling Analysis of ImageNet and FGSM)

Similar to the above analysis for Mnist, an image x is selected with original class "Rose-Hip", an adversarial perturbation x_a is prepared so that the sum $x + x_a$ has class "Baseball", and 50 Gaussian samples are generated for each of 1000 evenly spaced σ values. These samples are added to the above images and the classifications are colored via a heatmap:

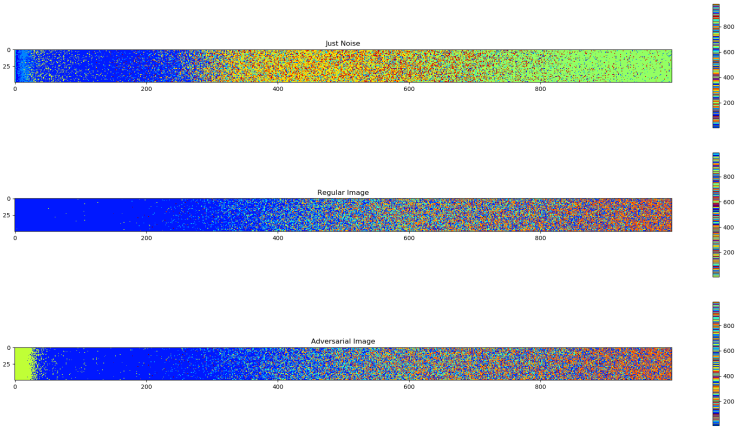


Figure 12: Top heatmap indicates Gaussian samples added to a pure black image,
Middle heatmap indicates Gaussian samples added to x (the original image),
Bottom heatmap indicates Gaussian samples added to $x + x_a$ (the adversarial image).

50 such examples are generated and bracketing is used to determine the σ for $(\gamma = 0.7, \sigma)$ stability in the following histogram. The adversarial example is denoted with a red dot.

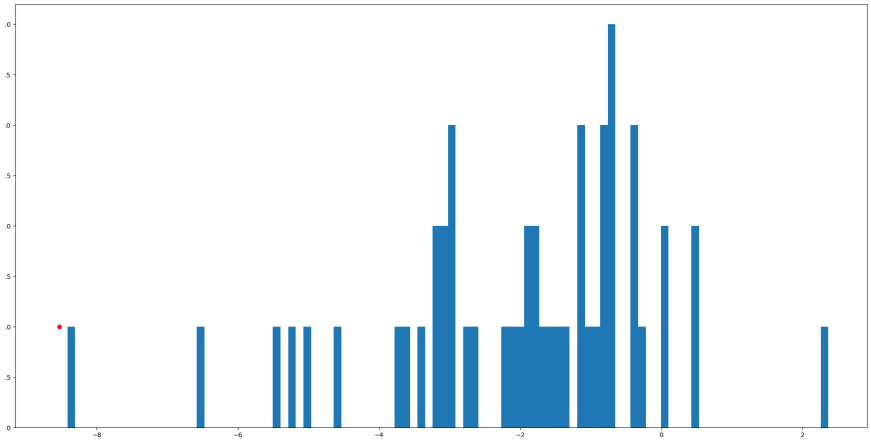


Figure 13: Histogram of σ for $(\gamma = 0.7, \sigma)$ stability of ImageNet examples with class "Baseball." Red dot on the left indicates the same σ measurement for an adversarial perturbation of a "Rose-Hip" with new class "Baseball" 6

Much like the above case for Mnist, IGSM examples generated on ImageNet were much less $\gamma - \sigma$ stable than natural images, again about 98% of adversarial images could be distinguished by their $\gamma - \sigma$ stability. It should be noted that due to computational intensity of processing ImageNet samples, this dataset is significantly smaller than that used for the Mnist experiments.

10.3 L-BFGS Neighborhood Sampling

In the following experiments, we will use the slower but more reliable L-BFGS technique from [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus] to prepare adversarial examples and will recreate results from the paper relating distortion with network robustness. The same networks and examples will then be subjected to $\gamma - \sigma$ stability analysis for $\gamma = 0.7$ to determine any correspondence of stability with network accuracy and average distortion.

10.3.1 Re-examining Szegedy Results with sampling

The following is a recreation of Table 1 from [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Good
in which networks of differing complexity are trained and attacked using L-BFGS. In addition to recreating the results from Szegedy et al. the table contains columns for $(\gamma = 0.7, \sigma)$ stability of natural and adversarial examples for each network. The networks listed are FC10-4, a fully connected network with no hidden layers that that takes the input vector $x \in \mathbb{R}^{784}$ to an output vector of $y \in \mathbb{R}^{10}$ with the $\lambda * \|\vec{w}\|_2 / N$ where $\lambda = 10^{-4}$ and N is the number of parameters in $\vec{w}$) added as regularization to the objective function during training. FC10-2 is the same except with $\lambda = 10^{-2}$ and FC10-0 has $\lambda = 1$ (a very large coefficient for regularization). FC100-100-10 and FC200-200-10 are networks with 2 hidden layers with regularization added for each layer of perceptrons with the λ for each layer equal to $10^{-5}, 10^{-5}$, and 10^{-6}. Training for these networks is conducted with a fixed number of epochs.

Network	Test Acc	Avg Distortion	Adversarial $(\gamma = 0.7, \sigma)$	Natural $(\gamma = 0.7, \sigma)$
FC10-4	92.09	0.123	1.68	0.93
FC10-2	90.77	0.178	4.25	1.37
FC10-0	86.89	0.278	12.22	1.92
FC100-100-10	97.31	0.086	0.56	0.65
FC200-200-10	97.61	0.087	0.56	0.73

Table 1: Recreation of Table 1 from Szegedy et al. using pytorch. New columns show σ values which achieved $(\gamma = 0.7, \sigma)$ stability for Adversarial and Natural examples.

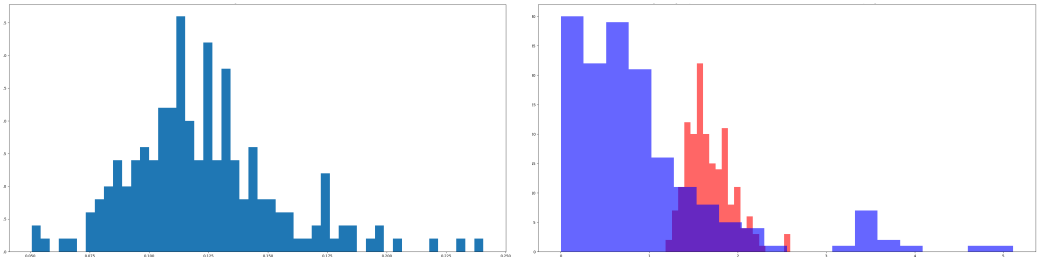


Figure 14: Distortion and σ histograms for FC-10-4 in Table 1

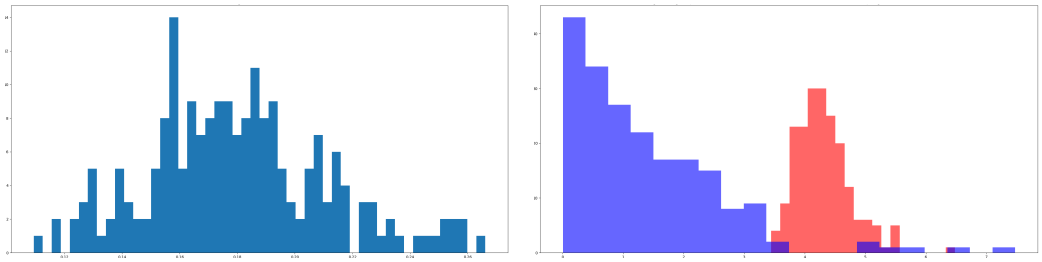


Figure 15: Distortion and σ histograms for FC10-2 in Table 1

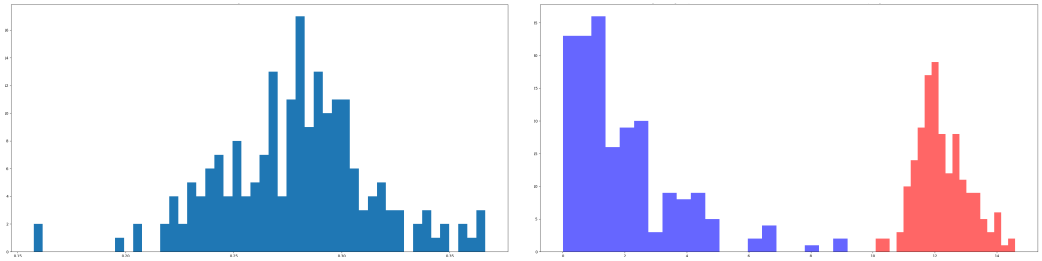


Figure 16: Distortion and σ histograms for FC10-0 in 1

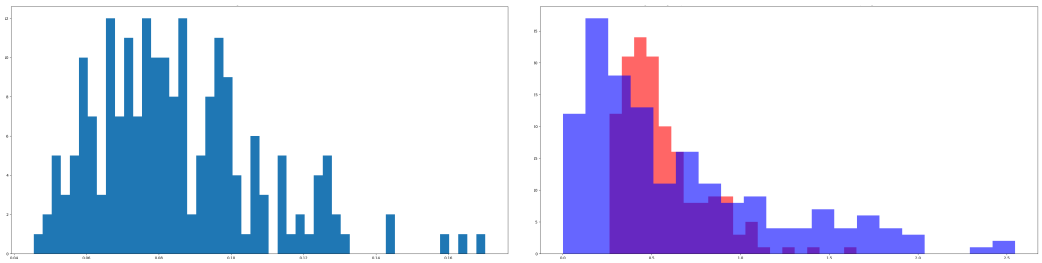


Figure 17: Distortion and σ histograms for FC100-100-10 in 1

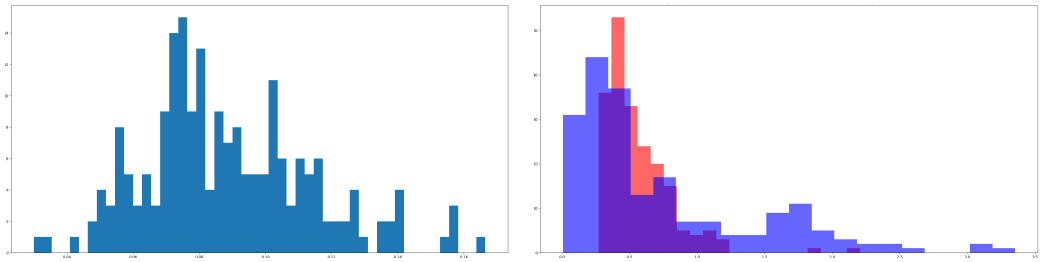


Figure 18: Distortion and σ histograms for FC200-200-10 in 1

10.4 Discussion of Experimental Results

In these results, we first notice that adversarial attacks generated with IGSM against Mnist are less stable than natural examples from these data sets. This difference is even more pronounced

when comparing a adversarial image with other image samples from its target class. This lower stability of adversarial examples is even more pronounced among the attacks prepared via IGSM against VGG16 on ImageNet. Using this technique, we can distinguish adversarial examples prepared by these techniques from natural examples using stability as a test.

In the next section of experiments, Table 1 from [Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, G was reconstructed and augmented with stability measurements. In this table, of the 5 networks trained on the Mnist dataset, the more complicated networks achieved higher accuracy but were defeated by adversarial examples with much less distortion than the simpler adversarial examples. We have added columns indicating the stability of natural and adversarial examples in these models. Adversarial examples generated for more complicated models were much less stable (some less stable than the corresponding natural examples). Adversarial examples generated for the less complicated networks were vastly more stable than natural examples. These results are summarized in the histograms Figure 14 through 18.

Part II
Decision Boundaries

11 decision boundary definitions

12 The Argmax Function

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

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

$$\text{argmax} : \mathbb{R}^k \to \mathcal{P}(C) \tag{7}$$

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

Restricting to only the pre-image of the singletons, it should be clear that argmax is continuous.

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

Further, we can directly prove that the pre-image of an individual singleton is open. Observe that for any point whose image is a singleton, one element of the domain vector must exceed the others by $\varepsilon > 0$. We shall use the ℓ^1 metric for distance, and thus if we restrict ourselves to a ball of radius ε, then all elements inside this ball will have that element still larger than the rest and thus map to the same singleton under argmax. Since the union of infinitely many open sets is open in $\mathbb{R}^k$, the union of all singleton pre-images is an open set. Conveniently this also provides proof that the union of all of the non-singleton sets in $\mathcal{P}(C)$ is a closed set. We will call this closed set the argmax Decision Boundary.

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

Questions:

Is the decision boundary connected

13 Defining Decision Boundaries

13.1 Complement Definition

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

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

13.2 Constructive Definition

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

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

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

13.3 Level Set Definition

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

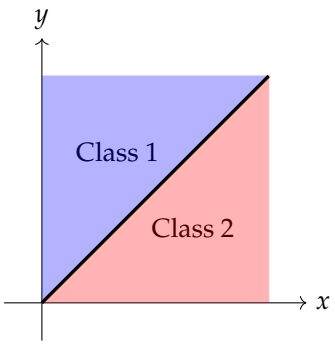
13.4 Images of Decision Boundaries

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

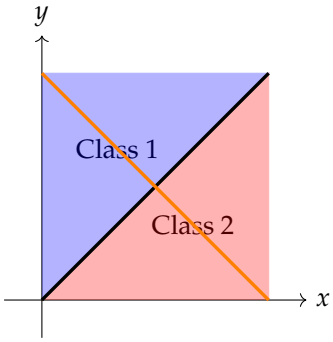
$$X \xrightarrow{F} Y \xrightarrow{\text{argmax}} \mathcal{C} \tag{8}$$

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

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



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



We note that the point $(0.5, 0.5)$ is therefore the only point on the decision boundary for probability valued F. We may generalize to higher dimensions where all probability valued models F will map into the the plane $x + y + z + \cdots = 1$ in Y and the decision boundary will be partitioned into $K - 1$ components, where the K-decision boundary is the intersection of this plane with the *centroid* line $x = y = z = \cdots$ and the 2-decision boundaries become planes intersecting at the same line.

14 Properties of Decision Boundaries

In practice, decision boundaries have a variety of properties. The decision boundary is a pre-image under an onto function that is not 1-1, which allows decision boundaries to vary in upto as many dimensions as the data space. In practice, this will likely be many fewer. In particular, decision boundaries should correspond with the structure of the data. If the data is embedded within a k-manifold within the input space, the decision boundary can be though of on the dual space of that manifold.

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

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

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

14.1 Delaunay Decision Boundaries

The simplest special case within this dual space is a classifier which is defined using the Voronoi diagram for the dataset. We can use the voronoi diagram to define a classifier for an arbitrary query point by simply assigning it the class of the training point whose voronoi cell contains it. This has several useful properties, though one in particular stands out.

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

Proof:

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

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

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

It is immediately clear that for points not in the training set, the line connecting them with eachother or with a point in the training set need not cross orthogonal to the voronoi cell boundaries. In order to talk clearly about such non-orthogonal crossings, we will define

Definition 14.1. *In arbitrary dimensional space, the* angle of incidence *between a line and a plane is defined as the angle between the line and its projection onto the plane, measured in the plane.*

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

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

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

and

Definition 14.2. *The*

14.2 Level Set Definition

Definition 14.3. *The* decision boundary *D of a probability valued function F is a union of all level sets $L_{c_1,c_2,a}$ defined by two classes c_1 and c_2 and a constant a which satisfy two properties: First, given $x \in L_{c_1,c_2,a}$, $F(x)_{c_1} = F(x)_{c_2} = a$ where a is a constant also defining the level set. Second, for all $c \notin \{c_1, c_2\}$, we have $a > F(x)_c$. This is the* decision level set *for level a. For a given point x on the decision boundary of a function, the* decision level *is the value a defining the level set of the decision boundary of which x is a member.*

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

Definition 14.4. *The* dimension *of the decision boundary at a point x is the number of dimensions needed to span $\{z : (x + z)$ is in the decision level a of the level set containing $x\}$*

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

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

14.3 Weighted Delaunay Decision Boundaries

This simple voronoi classifier has the limitation that it only cares about which individual training datapoint is closest to the query point, but is not sensitive to the distribution of this data. If we suppose that this data is distributed on a manifold with significant co-dimension with respect to the ambient data space, then it would be best to use a metric based on this manifold, so that distance among the data, and hence the delaunay triangulation and the comensurate voronoi cells would be determined by this metric and thus properly sensitive to the underlying geometry of this distribution. Alas, this is akin to having the generating function for nature. It cannot be computed. So instead, we can take advantage of the Delaunay triangulation with the usual euclidean metric as an approximation of the lower dimensional manifold in this case.

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

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

Definition 14.5. *Let M be a low dimensional manifold and let X_ε be a sampling of points on M for which the maximal edge length along its Delaunay triangulation D_ε is ε. Given a query point z, the $\varepsilon-$Delaunay weight for class i, $w(z)_i$, is the sum of the barycentric coordinates of z relative to the vertices with label i of the simplex of D_ε which contains z. The weight vector ε boundary level set for factor a, $B_{\varepsilon,a}$ is the level set $\{z \in M : \text{for all } i, \text{we have } w(z)_i \leq a \text{ and for some } i, j,\ w(z)_i = w(z)_j = a\}$. The union of all such level sets over a is the $\varepsilon-$*Delaunay-Boundary B_ε.

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

Theorem 14.6. *Given a low dimensional manifold M,*

$$\lim_{\varepsilon \to 0} B_\varepsilon(M) = B(M) \tag{9}$$

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

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

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

14.4 Orthant and Wedge

In order to study decision boundaries, we must talk about properties related to how decision boundaries interact. We desire a test-object whose dimensionality and acuteness of angle we can control. To this end we will use the $k-$dimensional orthant:

Definition 14.7. *Given an integer k, the k-*orthant *in n dimensions is the set* $\{x \in \mathbb{R}^n$ *such that for every natural number* $i \leq k$, $x_i > 0\}$

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

Definition 14.8. *Given integer k and a set of* $k-1$ *angles* Θ, *the* $k-\Theta-$wedge *in n dimensions is the set* $\{x \in \mathbb{R}^n$ *such that* $x_1 > 0$ *and for every natural number* $1 < i \leq k$, $x_i > x_1 \tan(\theta_{i-1})\}$.

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

Definition 14.9. *The* centerline $k-\Theta-$*wedge crossing is defined along the line* $x_1 = t$, *for* $1 < i \leq k$ $x_i = t\tan(\theta_{i-1}/2)$, *and for* $i > k$ $x_i = 0$.

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

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

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

Definition 14.10. *Given a finite set of half-planes* Y_i *in* $\mathbb{R}^n$ *and a point x, let I contain all i such that* $x \notin Y_i$. *Then since the intersection of any number of convex sets is convex,* $Y_x = \cap_{i \in I} Y_i$ *is a convex set. Furthermore, there is exactly one point z with* $|x - z| \leq |x - y|$ *for every y in* Y_x. *This is the* projection *of x onto* Y_x.

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

Lemma 14.11. *If all half-planes* Y_i *with corresponding normal vector* v_i *have boundaries* $B(Y_i)$ *which are mutually orthogonal, i.e. if* $\langle v_i, v_j \rangle = 0$ *for every combination of i and j, then the nearest point can be solved by iterative projection*

$$proj_{\mathbf{v_{i_1}}}(proj_{\mathbf{v_{i_2}}}(proj_{\mathbf{v_{i_3}}}...(\mathbf{x}))) \tag{10}$$

([**?**])

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

$$proj_{\mathbf{p_i}}(\mathbf{x}) = x - x \cdot v_i \tag{11}$$

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

$$\begin{aligned} w_i \cdot w_j &= \langle e_i \cos(\theta_i) - e_1 \sin(\theta_i), e_j \cos(\theta_j) - e_1 \sin(\theta_j) \rangle && (12)\\ &= e_i e_j \cos(\theta_i)\cos(\theta_j) - e_1 e_j \cos(\theta_j)\sin(\theta_j) - e_1 e_i \cos(\theta_i)\sin(\theta_j) + e_1^2 \sin(\theta_j)\sin(\theta_i) && (13)\\ &= e_1 \sin(\theta_j)\sin(\theta_i) && (14)\\ & && (15) \end{aligned}$$

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

$$\begin{aligned} v_1 \cdot w_i &= \langle e_1, -e_1 \sin(\theta_i) + e_i \cos(\theta_i) \rangle && (16)\\ &= -\sin(\theta_i) && (17) \end{aligned}$$

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

$$\begin{aligned} w_{ij} \cdot w_j &= 0 && (18)\\ w_{ij} &\perp \cap(w_i, w_j) && (19)\\ w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) &= 0 && (20) \end{aligned}$$

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

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

$$\begin{aligned} w_{ij} &= w_j + a = e_j \cos\theta_j - e_1 \sin\theta_j + a && (21)\\ a &= a_1 e_1 + a_2 e_2 + \cdots a_n e_n && (22)\\ w_{ij} \cdot w_i &= \sin\theta_j \sin\theta_i - a_1 \sin\theta_j + a_i \cos\theta_i = 0 && (23)\\ w_{ij} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) &= a_1 - a_i \tan\theta_i + a_j \tan\theta_j && (24) \end{aligned}$$

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

them all be 0. Then

$$-a_i \cos\theta_i = \sin\theta_j \sin\theta_i \tag{25}$$
$$a_i = -\sin\theta_j \tan\theta_i \tag{26}$$
$$a_j \tan\theta_j = \sin\theta_j \tan^2\theta_i \tag{27}$$
$$a_j = \cos\theta_j \tan^2\theta_i \tag{28}$$
$$w_{ij} = w_j - e_i \sin\theta_j \tan\theta_i + e_j \cos\theta_j \tan^2\theta_i \tag{29}$$

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

$$w_{ijk} = w_k + a = e_k \cos\theta_k - e_1 \sin\theta_k + a \tag{30}$$
$$a = a_1 e_1 + a_2 e_2 + \cdots a_n e_n \tag{31}$$
$$w_{ijk} \cdot w_i = \sin\theta_k \sin\theta_i - a_1 \sin\theta_k + a_i \cos\theta_i = 0 \tag{32}$$
$$w_{ijk} \cdot w_j = \sin\theta_k \sin\theta_j - a_1 \sin\theta_j + a_j \cos\theta_j = 0 \tag{33}$$
$$w_{ijk} \cdot (e_1 + e_i \tan\theta_i + e_j \tan\theta_j) = a_1 + a_i \tan\theta_i + a_j \tan\theta_j + a_k \tan\theta_k \tag{34}$$

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

$$-a_i \cos\theta_i = \sin\theta_k \sin\theta_i \tag{35}$$
$$a_i = -\sin\theta_k \tan\theta_i \tag{36}$$
$$-a_j \cos\theta_j = \sin\theta_k \sin\theta_j \tag{37}$$
$$a_j = -\sin\theta_k \tan\theta_j \tag{38}$$
$$a_k \tan\theta_k = \sin\theta_k \tan^2\theta_i + \sin\theta_k \tan^2\theta_j \tag{39}$$
$$a_k = \cos\theta_k (\tan^2\theta_i + \tan^2\theta_j) \tag{40}$$
$$w_{ij} = w_j - e_i \sin\theta_k \tan\theta_i \tag{41}$$
$$- e_j \sin\theta_k \tan\theta_j \tag{42}$$
$$+ e_k \cos\theta_k (\tan^2\theta_i + \tan^2\theta_j) \tag{43}$$

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

Now, performing orthogonal projection, we can orthogonally project in sequence for all half-planes not containing x except for v_1 since e_1 is not orthogonal to any of the wedge planes. Once we have projected onto all other violated planes, if the projection onto e_1 is negative, this will mean that the nearest point to x is the origin. For $i \neq 1$, all these projections are of the first form. $x - \sum_i x \cdot e_i$.

The first projection of the second form can be done by simple orthogonal projection, however in order to perform the next projection, we must project onto the intersection of the two half-planes without leaving the first plane. We'll need a new plane which is orthogonal to the first plane, but has the same intersection with the second plane. We can compute what is missing to make this projection orthogonal. We have

$$w_i \cdot w_j = \langle e_i \sin(\theta_i) - e_1 \cos(\theta_i), e_j \sin(\theta_j) - e_1 \cos(\theta_j) \rangle \tag{44}$$

$$= e_i e_j \sin(\theta_i) \sin(\theta_j) - e_1 e_j \sin(\theta_j) \cos(\theta_j) - e_1 e_i \sin(\theta_i) \cos(\theta_j) + e_1^2 \cos(\theta_j) \cos(\theta_i) \tag{45}$$

$$= e_1 \cos(\theta_j) \cos(\theta_i) \tag{46}$$

$$\tag{47}$$

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

15 exploring boundary curvature with Random Walks

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

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

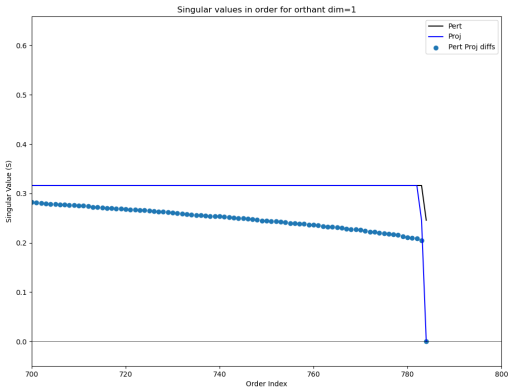
Initial comparison of SVDs of decision boundary points yields a single dominant singular value which corresponds with the content of the original image on the boundary. All random noise appears to be mostly orthogonal to this.

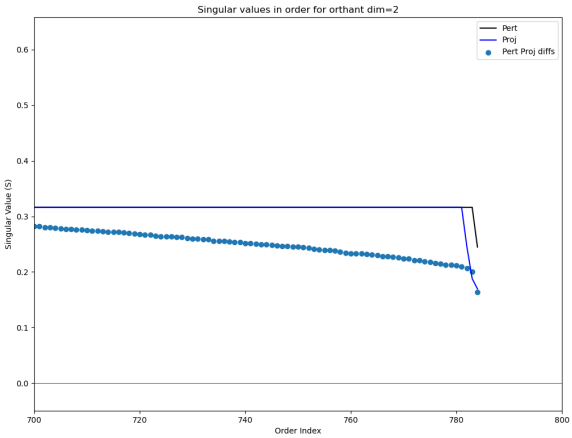
Once this vector is removed, the remaining signal is simply the adversarial attack information.

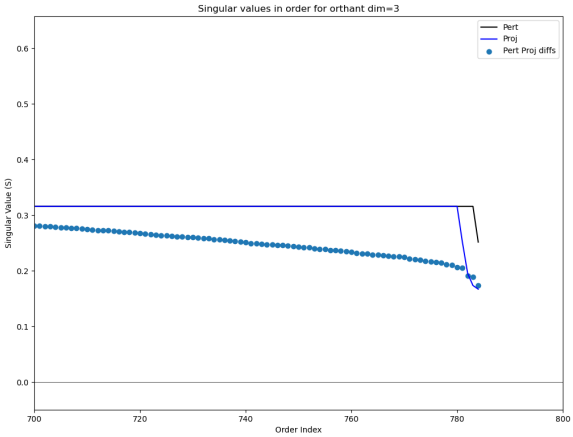
To get a set of singular vectors which to not emphasize the image content, we take random differences among the sampled images and take the SVD of those random differences.

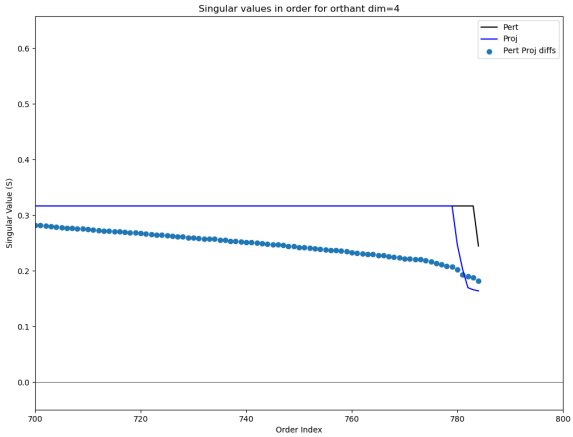
In these first 10 images, this procedure is carried out on the orthant, where samples are generated with mean 0 and are projected onto orthants with increasing numbers of dimensions. We can see a very clear dropped set of singular values smaller than the rest, which indicate the number of degenerate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

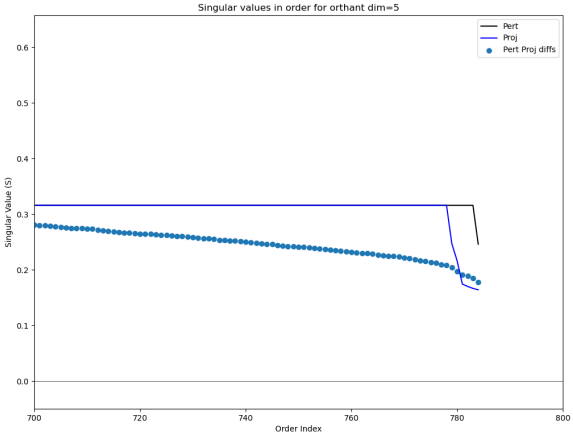
Experiment : A valid experiment here is to measure the difference between the images and the projection to get – in this case – normal vectors to the orthant for each image sampled. The SVD of these normals can be computed to determine the number of dimensions in the projecetion operation. This same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

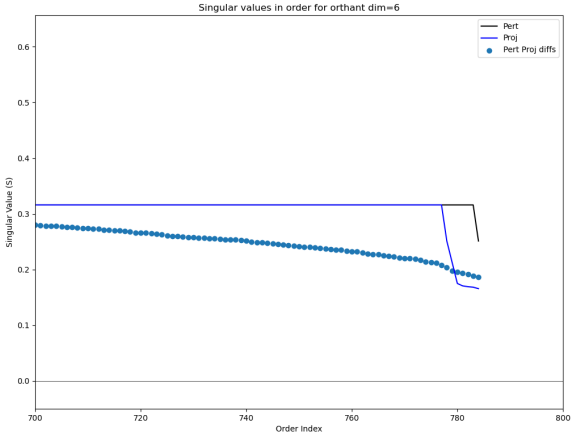

Singular values in order for orthant dim=1
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index

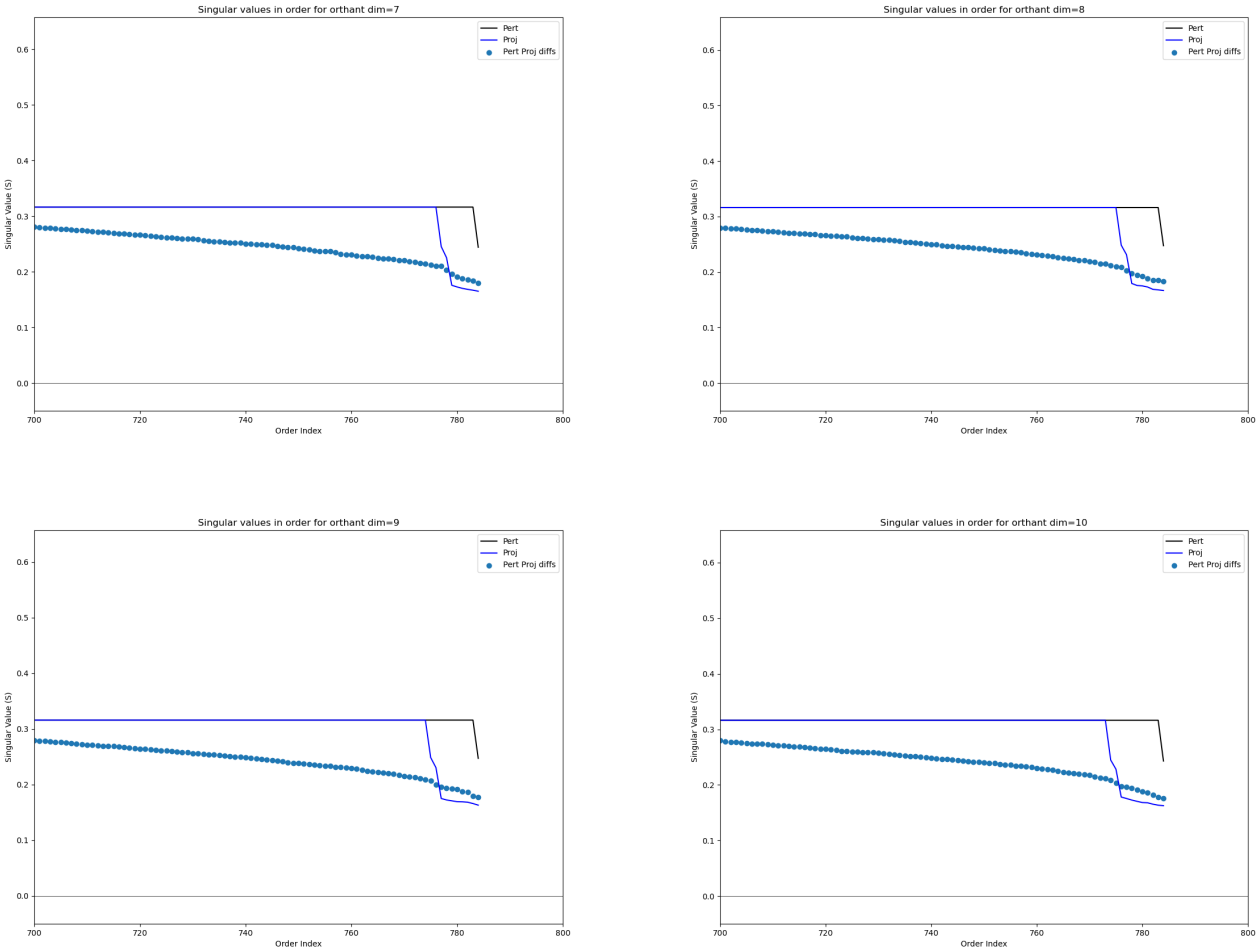

Singular values in order for orthant dim=2
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index


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


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


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

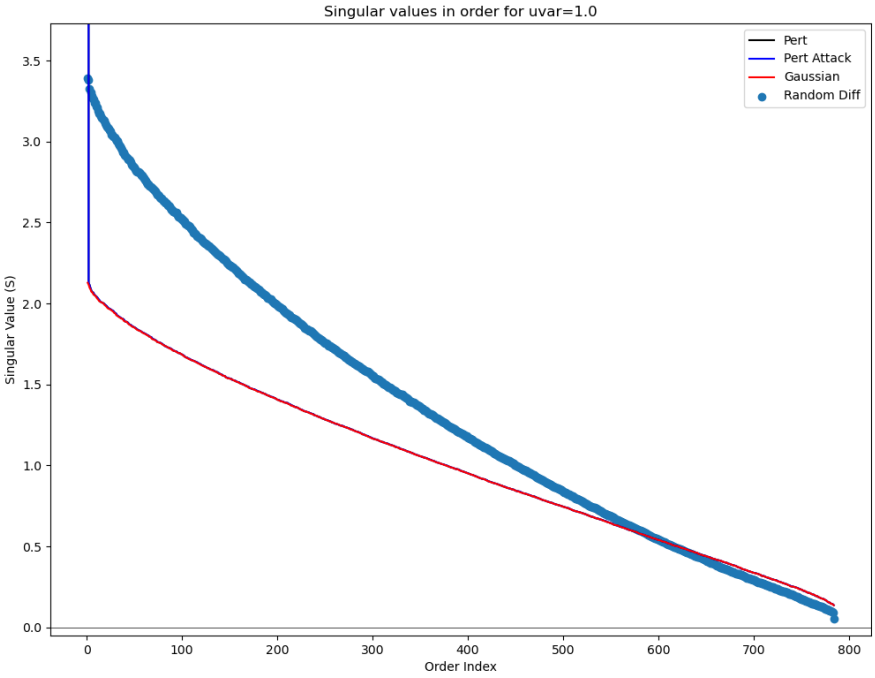

Singular values in order for orthant dim=6
Pert
Proj
Pert Proj diffs
Singular Value (S)
Order Index



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

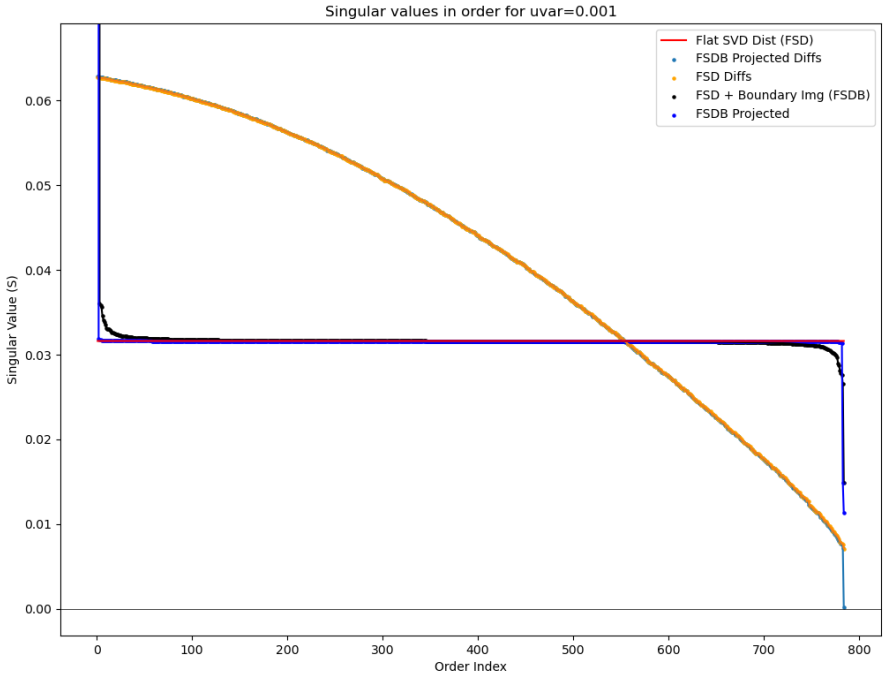
It remains to compare these computed normals with other known quantities, e.g. the gradient computed with respect to adversarial loss functions and the difference between each sampled image and its resultant projection. In addition, it remains to compare the normals of multiple nearby images to determine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

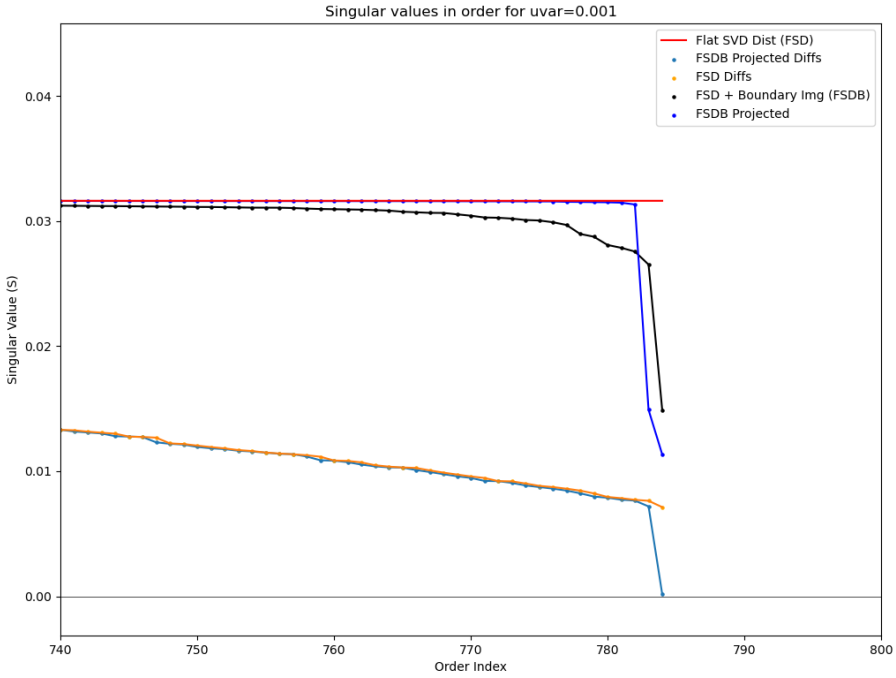
The following image shows singular values in order for a sample around a decision boundary image in MNIST generated by interpolating from a test image to an adversarial image generated from it. This plot is dominated by the natural distribution of singular values for a gaussian and also by the original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



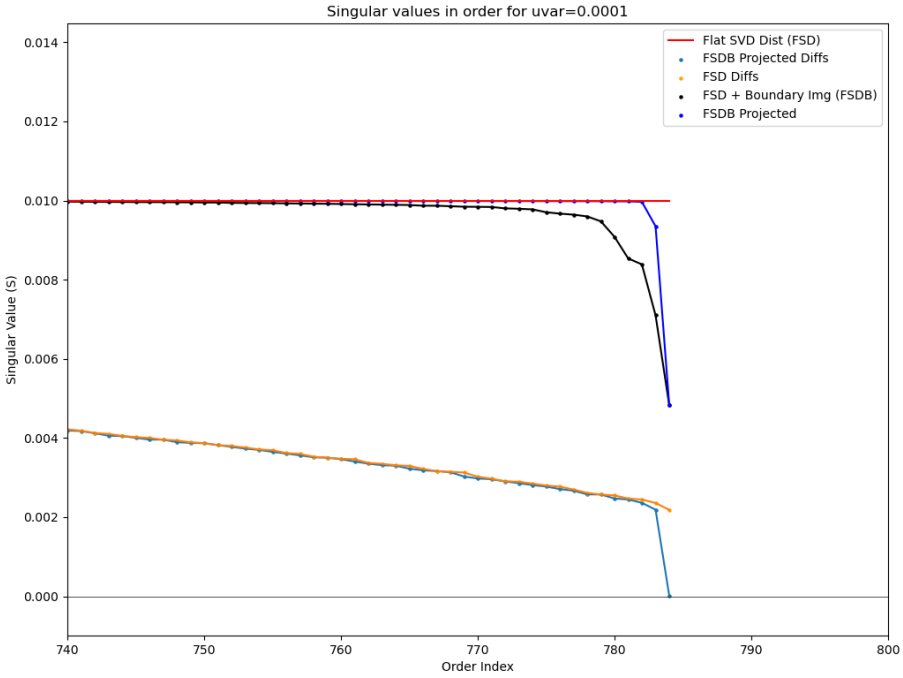
The following plots are generated with a new distribution to replace the gaussian. This "Flat SVD" distribution is generated by first taking a gaussian sample representable by a matrix X, then taking the SVD of this gaussian sample $U\Sigma V = X$, replacing Σ by $|\Sigma|_2 I$. This distribution has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

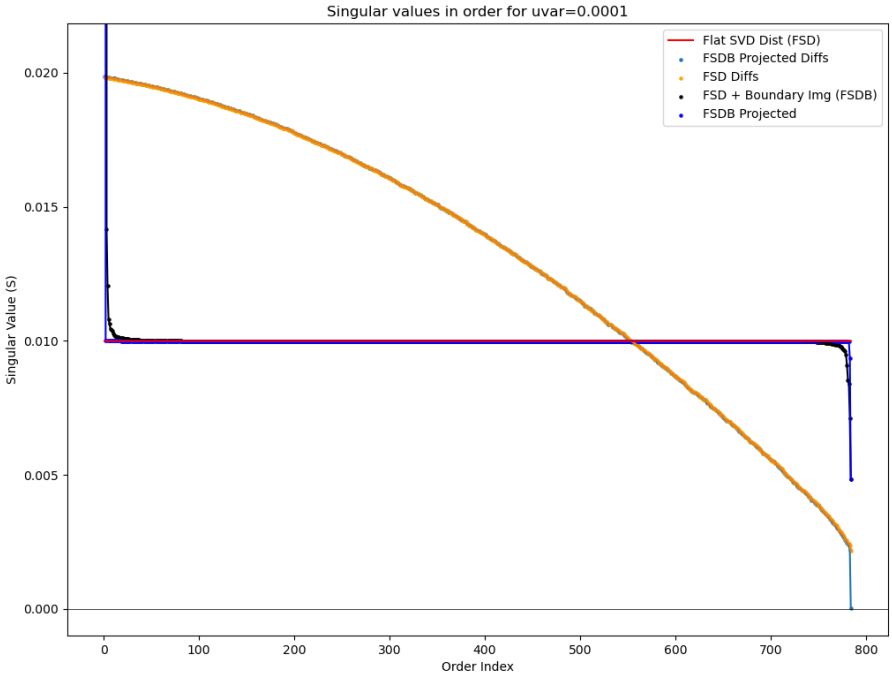
These first two plots are generated by sampling around a decision boundary image found by interpolating between two test images (rather than a test image and an adversarial example generated from it). We note that the SVD contains two degernate values. These seem to correpond with the two original images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


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



These last two images are generated with the flattened distribution

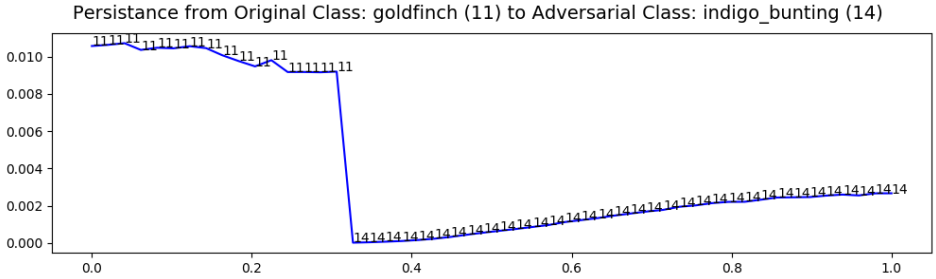

Singular values in order for uvar=0.0001
Flat SVD Dist (FSD)
FSDB Projected Diffs
FSD Diffs
FSD + Boundary Img (FSDB)
FSDB Projected
Singular Value (S)
Order Index
0.014
0.012
0.010
0.008
0.006
0.004
0.002
0.000
740
750
760
770
780
790
800



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

16 Re-Examining Persistence

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



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

We also wish to augment these tools to include

Abstract

In this work, we write an exact representation for an arbitrary gradient trained model as a kernel based method extending the theory of the path kernel described in [?] and discussed by [?]. We discuss the conditions under which this representation is exact, measure approximation error, and compare to the well known Neural Tangent Kernel (NTK) [?]. We implement this representation for an artificial neural network and demonstrate that it is computationally tractable and accurate in practice. Using this kernel, we quantify uncertainty according to this kernel using Gaussian process regression and discuss the implicit limitations that this reveals about neural networks. In particular, we show that the kernel resulting from a typical neural network is non-stationary and has highly unusual spatial properties.

25 Introduction

TODO rewrite to focus on the seconod paper.

This study investigates the relationship between kernel methods and models whose parameters are determined using gradient methods. The neural tangent kernel (NTK) is a well established method which represents training gradients as a kernel method [?] and compares trained models with this kernel in the case of models as their number of parameters approaches infinity. A theory

of the neural path kernel (NPK) has been proposed which integrates finite tangent kernels [**?**]. The path kernel extends from the NTK by directly constructing a kernel method approximation of a model by integrating tangent kernels along the continuous gradient flow defined by the model's gradient with respect to its training data and loss function. The case is made that this continuous path kernel is an exact representation of a continuous gradient trained model and that any practical model trained using discrete steps according to gradient descent is therefore approximately a kernel method. This opens up such models, including artificial neural networks (ANNs), to many theoretical tools available to kernel methods ([**?**, **?**, **?**]). However usage of such tools is highly dependent on the accuracy and dynamics of this approximation. Furthermore, the smooth path kernel requires difficult measurements of convergence and error in order to make practical comparisons with real discretely trained models. Although the convergence argued for likely holds as training step size converges to zero, these arguments do not help us understand the dynamics of this convergence and are very difficult to compute in practice.

In this work we propose an exact kernel representation for any gradient trained model satisfying very few conditions and demonstrate that this method is computable, practical, and exposes convergence and approximation properties to rigorous analysis. Our results demonstrate the potential for using the path kernel to study deep neural networks and provide a foundation for further research in this area. Furthermore, we will discuss relaxations of the required conditions on this kernel that allows very general functions to be represented with a small measurable approximation that can be bounded in practice. In addition, we study methods for approximating our discrete path kernel to reduce computation costs while maintaining many of these useful properties.

26 Related Work

[**?**] NTK

27 Discrete Path Kernels

Models trained by gradient descent can be characterized by a discrete set of intermediate states in the space of their parameters. These states are not bound to the gradient flow defined for such models, so we must consider how to integrate a discrete path for weights whose states differ from the gradient flow. In order to write this representation we must carefully define both kernel methods and kernels:

Definition 27.1. *A kernel is a function of two variables which is symmetric and positive definite.*

Definition 27.2. *Given a Hilbert space X, a query point $x \in X$, and a training set $X_T \subset X$, a* Kernel Method *is a model characterized by*

$$\hat{y}(x) = b + \sum_i a_i k(x, x_i) \tag{48}$$

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

By Mercer's theorem [**?**] 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 will 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.

Definition 27.3. *Let y_w be a differentiable function parameterized by $w \in \mathbb{R}^d$ which is trained via N forward Euler steps of fixed size ε on a finite subset $X_T = \{x_i\}_{i=1}^M$ of a Hilbert space X of size M with labels $Y_T = \{y_i\}_{i=1}^M$, with initial parameters w_0 so that there is a constant $b \in \mathbb{R}$ such that for all x, $\hat{y}_{w_0}(x) = b$, and weights at each step $w_s : 0 \leq s \leq N$. Let $x \in X$ be arbitrary and within the domain of $\hat{y}_w$ for every w. Then the* discrete path kernel *(DPK) can be written*

$$K_{DPK}(x, x') = \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x') \rangle dt \tag{49}$$

where

$$a_{i,s} = -\varepsilon \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \in \mathbb{R} \tag{50}$$

$$\phi_{s,t}(x) = \nabla_w \hat{y}_{w_s(t,x)}(x) \tag{51}$$

$$w_s(t, x) = \begin{cases} w_s, x \in X_T \\ w_s(t), x \notin X_T \end{cases} \tag{52}$$

Lemma 27.4. *The discrete path kernel (DPK) is a kernel.*

Theorem 27.5 (Exact Kernel Representation)**.** *A model $\hat{y}_{w_N}$ trained using discrete steps matching the conditions of the discrete path kernel has the following exact kernel method representation:*

$$\hat{y}_{w_N}(x) = b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} K_{DPK}(x, x') \tag{53}$$

Remark 0 We can see that by changing equation 53 we can produce an exact representation for any discrete optimization scheme that can be written in terms of model gradients. This could include backward Euler, leapfrog, higher order schemes (which are generally intractable for Artificial Neural Networks), and any variation of adaptive step sizes.

Remark 1 $\phi_{s,t}(x)$ 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 discrete path compared with the gradient flow for each step. We can write another function k' which is continuous but not symmetric, but still produces an exact representation:

$$k'(x, x') = \langle \nabla_w \hat{y}_{w_s(t)}(x), \nabla_w \hat{y}_{w_s(0)}(x') \rangle \tag{54}$$

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

$$\int_0^1 \nabla_w \hat{y}_{w_s(t)}(x) dt = \nabla_w \hat{y}_{w_s(0)}(x) \tag{55}$$

The asymmetry of this function is exactly measuring the disagreement between the discrete steps taken during training with the gradient field defined by the loss function composed with the model. This function is one of several subjects for further study, particularly in the context of gaussian processes whereby the asymmetric matrix corresponding with this function can stand in for a covariance matrix. It may be that gaussian thermostats can be used to repair they asymmetry while accounting for any loss accrued due to disagreement between the discrete steps and the gradient flow or it may be that the not-symmetric analogue of the covariance in this case has physical meaning relative to uncertainty.

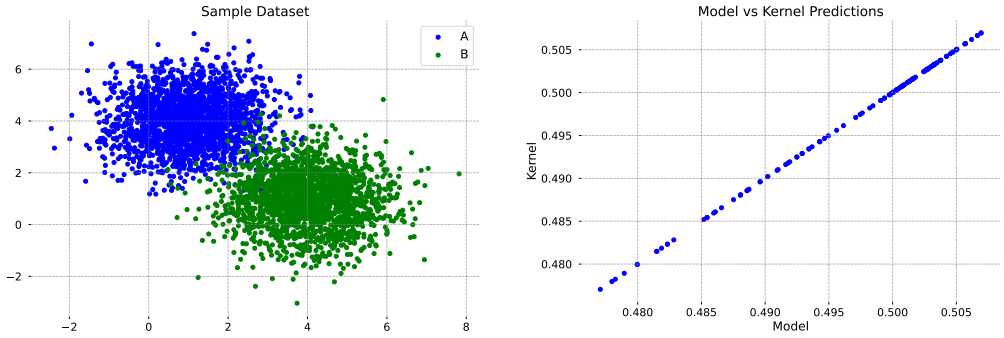


Figure 19: On the left is a 2d dataset of points sampled from gaussians with different means. Specifically, class A is normally distributed with $\mu = [1, 4]$ and $\sigma^2 = 1$ while class B is $\mu = [4, 1]$ and $\sigma^2 = 1$. 2000 data points were sampled for each class. These values were chosen arbitrarily to provide separation with a limited amount of overlap. On the right is the prediction similarity between the kernel and the original model. This demonstrates that our kernel formulation accurately represents the trained network.

Remark 2 In order to obtain an exact representation, one must start with a model that has constant output for all input, i.e. for every x and $\hat{y}_0(x) = b$ (e.g. an ANN with all weights in the final layer initialized to 0). When relaxing this property, to allow for models that have a non-constant starting output, we note that this representation ceases to be exact. The resulting approximate representation will still agree strongly with the ANN, and will converge quickly in output as the ratio of the length of the training path divided by the step size goes to infinity. In fact this convergence is very rapid and useful approximation will be achieved within typical training length. (TODO : lipshitz argument)

Remark 3 We note that since f is being trained using forward Euler, we can write:

$$\frac{\partial w_s(t)}{dt} = -\varepsilon \nabla_w L(\hat{y}_{w_s}(x_i), y_i) = -\varepsilon \sum_{j=1}^{d} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial w_j} \tag{56}$$

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

$$\int_0^1 \nabla_w \hat{y}_{w_{s,\varepsilon}(t)}(x) dt \to \nabla_w \hat{y}_{w_s(0)}(x) \tag{57}$$

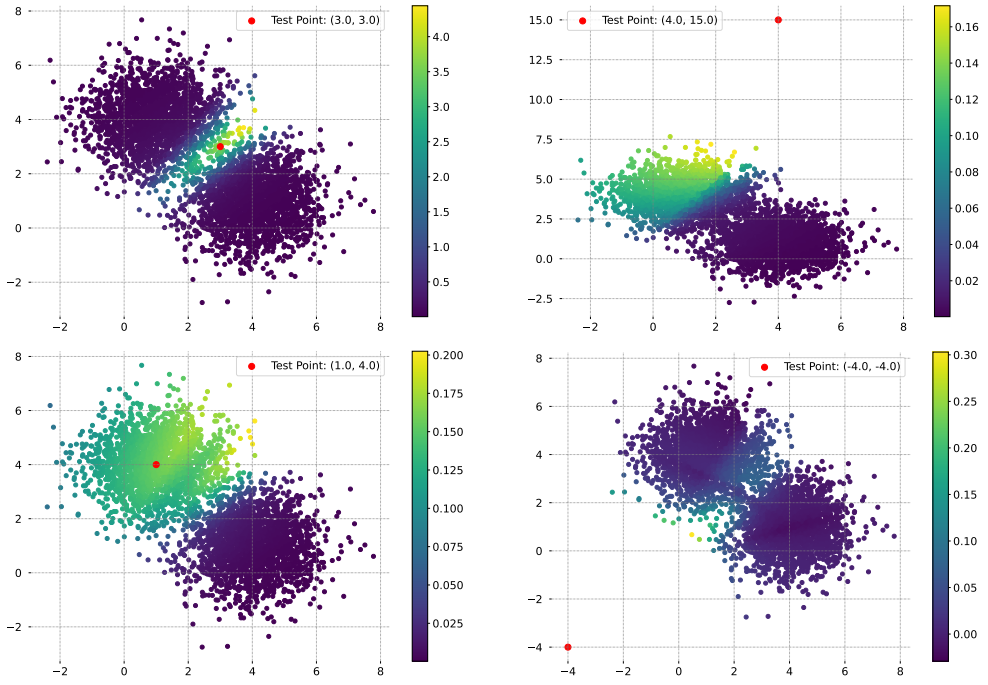
In particular, given a model $\hat{y}$ that admits a Lipshitz constant K this approximation has error bounded by εK and a proof of this convergence is direct.

28 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.

28.1 Evaluating The Kernel

Examples of the kernel values across 4 test points are shown in Figure 20. We are interested in how the kernel is learning and whether this kernel will allow out-of-distribution (OOD) detection.



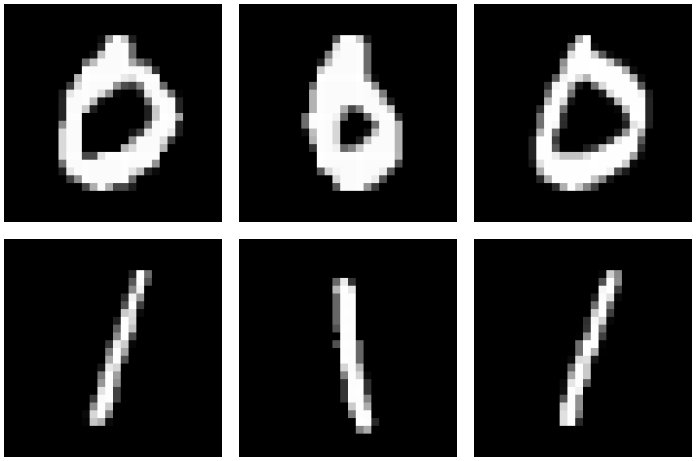
In-Distribution Out-Of-Distribution

Figure 20: Example of the kernel values on in-distribution and out-of-distribution (OOD) data. Left column shows samples which are in-distribution for our dataset. Right column row shows OOD samples.

From these plots we see that the in-distribution samples have a significantly higher sum over kernel distances than the OOD examples. Of note is that the OOD detection is not perfect. For the test point $(4.0, 15.0)$ it still identifies a large portion of class A samples as being relatively close in kernel space. Despite this, both OOD examples shown are significantly lower in total kernel distance than the in-distribution samples. Further experiments will be required to better understand why some OOD regions are closer than others

28.2 Extending To Image Data

We perform experiments on MNIST to demonstrate the applicability to image data. This kernel representation was generated for a two-layer fully connected ReLU Network with the cross-entropy loss-function, using Pytorch (citation). The model was trained using forward Euler (gradient descent) using gradients generated as a sum over all training data for each step. The state of the model was saved for every training step. In order to compute the per-training-point gradients needed for the kernel representation, the per-input jacobians are computed at execution time in the representation by loading the model for each training step i , computing the jacobians for each training input to compute $\nabla_w \hat{y}_{w_s(0)}(x_i)$, and then repeating this procedure for 100 t values between 0 and 1 in order to approximate $\int_0^1 \hat{y}_{w_s(t)}(x)$. Torch is not currently optimized to provide per-input jacobians or to provide jacobians for multiple weight states, so this procedure can be accelerated greatly by deeper integration with existing pytorch tools.



Original Image Kernel Distance Pixel Distance

Figure 21: Comparison between the nearest samples in kernel space and pixel space. From left to right in each column: Test set point, nearest sample in kernel space, nearest sample in pixel space using euclidean distance.

We are able to see the agreement between the neural network and its kernel representation in figure 22. In figure 21 we see that the function learned by the kernel does not directly mimic euclidean distance in the image space. Samples which are nearby in kernel space are not necessarily nearby in pixel space. The similarity metric learned is a direct explanation of how the neural network is making decisions.

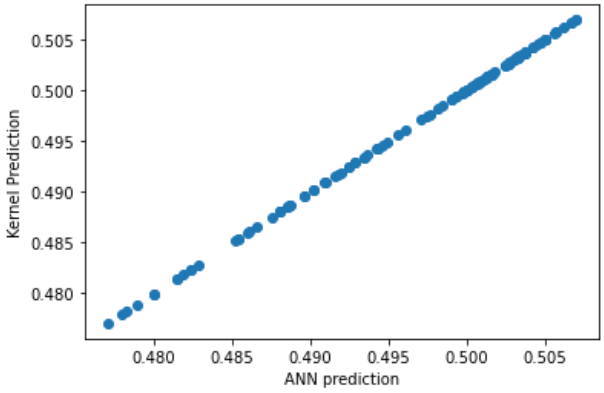


Figure 22: This plot shows output of the ANN versus output of the corresponding kernel representation for a set of test images from the MNIST dataset. We note the very strong agreement between the two outputs.

29 Discussion

The implications of a practical and finite kernel representation for neural networks are wide and profound. For most gradient trained models, there is a disconnect between the problem space (e.g. images) and the parameter space of a network. Parameters are intrinsically un-interpretable and much work has been spent building approximate mappings that convert model understanding back into the problem space in order to interpret features, sample importance, and other details ([?], [?], and [?]). A kernel is composed of a direct mapping from the problem space into parameter space. This mapping allows much deeper understanding of gradient trained models because the internal state of the method has an exact representation mapped from the problem space. Sample importance is produced directly by looking at the kernel and its corresponding weights per training input.

As stated in previous work [?], this representation has strong implications about the structure of gradient trained models and how they can understand the problems that they solve. Since the kernel weights in this representation are fixed derivatives with respect to the loss function L, $a_{i,s} = -\varepsilon \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i}$ nearly all of the information used by the network is represented by the kernel mapping function and inner product. Inner products are not just measures of distance, they also measure angle. In fact, figure 23 shows that for a typical training example, the L_2 norm of the weights changes monotonically by only 20-30% during training. This means that the "learning" of a gradient trained model is dominated by change in angle, which is predicted for kernel methods in high dimensions [?].

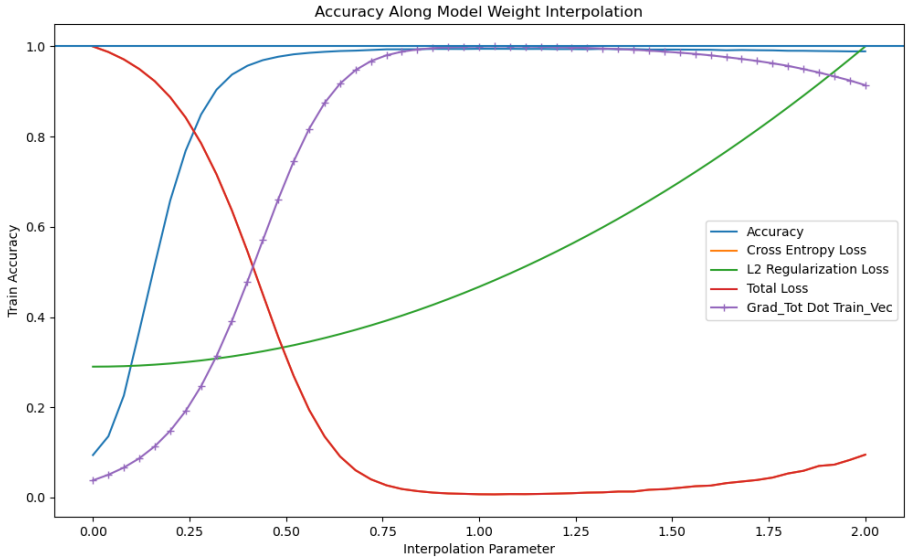


Figure 23: This plot shows a linear interpolation $w(t) = w_0 + t(w_1 - w_0)$ of model parameters w for a convolutional neural network $\hat{y}_w$ from their starting random state w_0 to their ending trained state w_1. The hatched blue line shows the dot product of the aggregated gradient over the training data X, $\langle \nabla_w \hat{y}_{w(t)}(X), (w_1 - w_0)/|w_1 - w_0| \rangle$. The other lines indicate accuracy (blue), total loss (red decreasing), and L2 Regularization (red increasing)

Perhaps the most significant advantage for gradient trained models of an exact kernel representation is that the combination of kernel and kernel weights provides a spatial representation of the model's understanding relative to the training data. In previous work ([?] [?] it has been shown

that image classification can be represented by projection onto the convex hull of training data. This projection is computationally infeasible, but it provides a geometric gold-standard classifier. Since kernel methods provide a spatial representation of their prediction, this representation can be directly compared with convex hull projections. It also provides data through which we can infer the gradient model's understanding of the data spatially.

For kernel methods, this also represents a significant step. Despite their firm mathematical foundations, kernel methods have lost ground since the early 2000s due to the limitations of developing new kernels for complex high-dimensional problems [**?**]. This opens up many modern problems to the powerful tools available to kernel methods. Of these, Gaussian Processes (GPs) may be the most exciting. Given our kernel function, we can generate covariance matrices for GP which will allow direct uncertainty measurement. This will allow much more significant analysis for out-of-distribution samples including adversarial attacks ([**?**] [**?**]).

Gaussian processes are one of many research directions that naturally follow this work. It is worth noting that since the kernel from our representation can be either continuous or symmetric but not both (See remark 27), covariance matrices used in GPs will have slightly unusual properties that will reflect the divergence of the discrete training path from the smooth gradient flow. In the case of the asymmetric version of our representation, this asymmetry measures this divergence in a way that may be tractably explored using gaussian thermostats ([**?**], [**?**]). In addition to exploring GPs, it is natural to pursue more efficient computation of these representations by exploiting features of pytorch and incorporating the necessary integral computation with some codes that have been developed for NeuralODEs which require similar information ([**?**], [**?**])

Another implication from this representation is the increased importance of models following their gradient flow during training. Since this derived kernel is either discontinuous or asymmetric depending on the neural network's training trajectory, developing training restrictions which satisfy equation 55 may produce more useful kernels and have implications about the accuracy and generalizability of ANN models. This will provide a new motivation for such research separate from just the question of efficiency of training. Approaches in this direction may be found in control theory ([**?**]) and the neural ODE approach ([**?**], [**?**]). Also in this vein is the precise formulation of the divergence error from the discrete training path to the smooth gradient flow. Such a formulation should shed light on the dynamics of how such representations converge in performance under various step refinements.

30 Acknowledgements

This research was funded by Los Alamos National Lab LDRD-DR XX9C UQ4ML (help with how to acknowledge this LDRD funding juston?) Thanks to Yen Ting Lin, Philip Hoskins, Keenan Eikenberry, and Craig Thompson for feedback on early iterations of this paper.

References

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

[Carlini and Wagner(2016)] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. *arXiv:1608.04644 [cs]*, Aug. 2016. URL http://arxiv.org/abs/1608.04644. arXiv: 1608.04644.

[Glorot et al.(2011)Glorot, Bordes, and Bengio] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse

rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[Goodfellow et al.(2013)Goodfellow, Bulatov, Ibarz, Arnoud, and Shet] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2013.

[Goodfellow et al.(2014)Goodfellow, Shlens, and Szegedy] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, Dec. 2014. URL http://arxiv.org/abs/1412.6572. arXiv: 1412.6572.

[Hardt et al.(2015)Hardt, Recht, and Singer] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240, 2015. URL http://arxiv.org/abs/1509.01240.

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

[Kak(1993)] S. Kak. On training feedforward neural networks. *Pramana*, 40(1):35–42, 1993.

[Khoury and Hadfield-Menell(2018)] M. Khoury and D. Hadfield-Menell. On the geometry of adversarial examples. *CoRR*, abs/1811.00525, 2018. URL http://arxiv.org/abs/1811.00525.

[Krause(2020)] A. Krause. Introduction to machine learning, August 2020.

[Kurakin et al.(2016)Kurakin, Goodfellow, and Bengio] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, July 2016. URL http://arxiv.org/abs/1607.02533. arXiv: 1607.02533.

[LeCun et al.(1988)LeCun, Touresky, Hinton, and Sejnowski] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

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

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

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

[Li and Yuan(2017)] Y. Li and Y. Yuan. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*, pages 597–607, 2017.

[Linnainmaa(1970)] S. Linnainmaa. 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, 1970.

[Liu and Deng(2015)] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*, pages 730–734. IEEE, 2015.

[Malik and Perona(1990)] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *JOSA A*, 7(5):923–932, 1990.

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

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

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

[Moosavi-Dezfooli et al.(2015)Moosavi-Dezfooli, Fawzi, and Frossard] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. *arXiv:1511.04599 [cs]*, Nov. 2015. URL http://arxiv.org/abs/1511.04599. arXiv: 1511.04599.

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

[Papernot et al.(2015)Papernot, McDaniel, Jha, Fredrikson, Celik, and Swami] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. *arXiv:1511.07528 [cs, stat]*, Nov. 2015. URL http://arxiv.org/abs/1511.07528. arXiv: 1511.07528.

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

[Prakash et al.(2018)Prakash, Moran, Garber, DiLillo, and Storer] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. A. Storer. Deflecting adversarial attacks with pixel deflection. *CoRR*, abs/1801.08926, 2018. URL http://arxiv.org/abs/1801.08926.

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

[Rumelhart et al.(1986)Rumelhart, Hinton, and Williams] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

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

[Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL http://arxiv.org/abs/1312.6199.

[Werbos(1974)] P. J. Werbos. 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, 1974.

[Wiyatno and Xu(2018)] R. Wiyatno and A. Xu. Maximal jacobian-based saliency map attack. *CoRR*, abs/1808.07945, 2018. URL http://arxiv.org/abs/1808.07945.

A Appendix

ex

A.1 The DPK is a Kernel

Lemma A.1. *The discrete path kernel (DPK) is a kernel.*

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

To see that K_{DPK} is **Positive Semi-Definite**, let $f = (f_1, f_2, \ldots, f_n)^\top \in \mathbb{R}^n$ be any vector. We need to show that $f^\top K_{\text{DPK}} f \geq 0$. We have

$$f^\top K_{\text{DPK}} f = \sum_{i=1}^n \sum_{j=1}^n f_i f_j \int_0^1 \langle \phi_{s,t}(x_i), \phi_{s,t}(x_j)\rangle dt \tag{58}$$

$$= \sum_{i=1}^n \sum_{j=1}^n f_i f_j \int_0^1 \langle \nabla_w \hat{y}_{w_s(t,x_i)}, \nabla w \hat{y}_{w_s(t,x_j)}\rangle dt \tag{59}$$

$$= \int_0^1 \sum_{i=1}^n \sum_{j=1}^n f_i f_j \langle \nabla_w \hat{y}_{w_s(t,x_i)}, \nabla_w \hat{y}_{w_s(t,x_j)}\rangle dt \tag{60}$$

$$= \int_0^1 \sum_{i=1}^n \sum_{j=1}^n \langle f_i \nabla_w \hat{y}_{w_s(t,x_i)}, f_j \nabla_w \hat{y}_{w_s(t,x_j)}\rangle dt \tag{61}$$

$$= \int_0^1 \langle \sum_{i=1}^n f_i \nabla_w \hat{y}_{w_s(t,x_i)}, \sum_{j=1}^n f_j \nabla_w \hat{y}_{w_s(t,x_j)}\rangle dt \tag{62}$$

Re-ordering the sums so that their indices match, we have (63)

$$= \int_0^1 \left\| \sum_{i=1}^n f_i \nabla_w \hat{y}_{w_s(t,x_i)} \right\|^2 dt \tag{64}$$

$$\geq 0, \tag{65}$$

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

Remark In the case that our mapping function φ is not symmetric, after re-ordering, we still

yield something of the form:

$$= \int_0^1 \left\| \sum_{i=1}^{n} f_i \nabla_w \hat{y}_{w_s(t,x_i)} \right\|^2 dt \tag{66}$$

$$\tag{67}$$

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

A.2 The DPK is an Exact Representation

Theorem 27.5 (Exact Kernel Representation)**.** *A model $\hat{y}_{w_N}$ trained using discrete steps matching the conditions of the discrete path kernel has the following exact kernel method representation:*

$$\hat{y}_{w_N}(x) = b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} K_{DPK}(x, x') \tag{53}$$

Proof. Let $\hat{y}_w$ 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, $\hat{y}_{w_0}(x) = b$, and weights at each step $w_s : 0 \leq s \leq N$. Let $x \in X$ be arbitrary and within the domain of $\hat{y}_w$ for every w. For the final trained state of this model $\hat{y}_{w_N}$, let $y = \hat{y}_{w_N}(x)$.

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

$$\frac{\partial w_s(t)}{dt} = (w_{s+1} - w_s) \tag{68}$$

$$\int \frac{\partial w_s(t)}{dt} dt = \int (w_{s+1} - w_s) dt \tag{69}$$

$$w_s(t) = w_s + t(w_{s+1} - w_s) \tag{70}$$

$$\tag{71}$$

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

$$\frac{\partial w_s(t)}{dt} = -\varepsilon \nabla_w L(\hat{y}_{w_s}(x_i), y_i) = -\varepsilon \sum_{j=1}^{d} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial w_j} \tag{72}$$

Applying chain rule and the above substitution, we can write

$$\frac{\partial \hat{y}}{dt} = \frac{d\hat{y}_{w_s(t)}}{dt} = \sum_{j=1}^{d} \frac{\partial \hat{y}}{\partial w_j} \frac{\partial w_j}{dt} \tag{73}$$

$$= \sum_{j=1}^{d} \frac{\partial \hat{y}_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \frac{\partial L(\hat{y}_{w_s}(X_T), Y_T)}{\partial w_j} \right) \tag{74}$$

$$= \sum_{j=1}^{d} \frac{\partial \hat{y}_{w_s(t)}(x)}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \frac{\partial \hat{y}_{w_s}(x_i)}{\partial w_j} \right) \tag{75}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \sum_{j=1}^{d} \frac{\partial \hat{y}_{w_s(t)}(x)}{\partial w_j} \frac{\partial \hat{y}_{w_s}(x_i)}{\partial w_j} \tag{76}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \nabla_w \hat{y}_{w_s(t)}(x) \cdot \nabla_w \hat{y}_{w_s}(x_i) \tag{77}$$

$$\tag{78}$$

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

$$y_{s+1} - y_s = \int_0^1 -\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \nabla_w \hat{y}_{w_s(t)}(x) \cdot \nabla_w \hat{y}_{w_s}(x_i) dt \tag{79}$$

$$= -\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \left(\int_0^1 \nabla_w \hat{y}_{w_s(t)}(x) dt \right) \cdot \nabla_w \hat{y}_{w_s}(x_i) \tag{80}$$

$$\tag{81}$$

For all N training steps, we have

$$y_N = b + \sum_{s=1}^{N} y_{s+1} - y_s \tag{82}$$

$$y_N = b + \sum_{s=1}^{N} -\varepsilon \sum_{i=1}^{M} \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \left(\int_0^1 \nabla_w \hat{y}_{w_s(t)}(x) dt \right) \cdot \nabla_w \hat{y}_{w_s}(x_i) \tag{83}$$

$$= b + \sum_{i=1}^{M} \sum_{s=1}^{N} -\varepsilon \frac{\partial L(\hat{y}_{w_s}(x_i), y_i)}{\partial \hat{y}_i} \int_0^1 \left\langle \nabla_w \hat{y}_{w_s(t,x)}(x), \nabla_w \hat{y}_{w_s(t,x_i)}(x_i) \right\rangle dt \tag{84}$$

$$= b + \sum_{i=1}^{M} \sum_{s=1}^{N} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{85}$$

Since an integral of a symmetric positive semi-definite function is still symmetric and positive-definie and likewise for discrete sums, this represention is a kernel method.

□

A.3 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 $sign(a_{i,s}) = sign(a_{i,0})$ for all $a_{i,s}$. In this case it is possible to let the sample weights of our final kernel machine equal $sign(a_{i,0})$. In order to show this, we impose some structure on the loss function and network. Here we show this is the case for binary crossentropy on a network with sigmoid activations on the logits. (TODO: More argument here using mercer's theorem. All positive linear combinations of kernels are kernels. There are cases where some negative coefficients are allowed but that's going to take a lot more thought. How do we extend this to say $aKa > 0$ for all a?)

Proof.

$$L(\hat{y}_i, y_i) = -y_i \ln(\hat{y}_i) - (1 - y_i) \ln(1 - \hat{y}_i) \tag{86}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = \frac{y_i - \hat{y}_i}{(\hat{y}_i - 1)\hat{y}_i} \tag{87}$$

For a binary classification problem it is standard to have $y_i \in \{0, 1\}$ and using a sigmoid activation on the final layer we have $\hat{y}_i \in (0, 1)$.

Assume $y_i = 0$.

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = \frac{0 - \hat{y}_i}{(\hat{y}_i - 1)\hat{y}_i} \tag{88}$$

$$= \frac{-1}{\hat{y}_i - 1} \tag{89}$$

$$= \frac{1}{|\hat{y}_i - 1|} \tag{90}$$

The last equality relies on the fact that $\hat{y}_i < 1$.

$$y_i = 0 \implies \frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} > 0 \tag{91}$$

Assume $y_i = 1$.

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = \frac{1 - \hat{y}_i}{(\hat{y}_i - 1)\hat{y}_i} \tag{92}$$

$$= \frac{1 - \hat{y}_i}{-(1 - \hat{y}_i)\hat{y}_i} \tag{93}$$

$$= -\frac{1}{\hat{y}_i} \tag{94}$$

Because $\hat{y}_i > 0$.

$$y_i = 1 \implies \frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} < 0 \tag{95}$$

This shows that the sign of the gradient of the loss function depends only on the label y_i, not on the predicted value of our model $\hat{y}_i$ and is constant through training. Therefore:

$$y_S = b - \varepsilon \sum_{i=1}^{N} \sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{96}$$

$$= b - \varepsilon \sum_{i=1}^{N} sign(a_{i,0}) \sum_{s=1}^{S} |a_{i,s}| \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{97}$$

This formulates a kernel machine where

$$a_{i,0} = sign(\frac{\partial L(\hat{y}_{w_0}(x_i), y_i)}{\partial \hat{y}_i}) \in \{-1, 1\} \tag{98}$$

$$K(x, x_i) = \sum_{s=1}^{S} |a_{i,s}| \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle \, dt \tag{99}$$

$$\phi_{s,t}(x) = \nabla_w \hat{y}_{w_s(t,x)}(x) \tag{100}$$

$$w_s(t, x) = \begin{cases} w_s, x \in X_T \\ w_s(t), x \notin X_T \end{cases} \tag{101}$$

$$b = 0 \tag{102}$$

□

This argument does not hold in the simple case of linear regression. Assume our loss is instead squared error. Our labels are continuous on $\mathbb{R}$ and our activation is the identity function.

$$L(\hat{y}_i, y_i) = (y_i - \hat{y}_{i,s})^2 \tag{103}$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i} = 2(y_i - \hat{y}_{i,s}) \tag{104}$$

This quantity is dependent on $\hat{y}_i$ and its sign is changing throughout training. (TODO: Make this more formal and rigorous)

In order for

$$\sum_{s=1}^{S} a_{i,s} \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{105}$$

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

$$= \sum_{s=1}^{S} 2(y_i - \hat{y}_{i,s}) \int_0^1 \langle \phi_{s,t}(x), \phi_{s,t}(x_i) \rangle dt \tag{106}$$

$$= \sum_{s=1}^{S} 2(y_i - \hat{y}_{i,s}) \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt \tag{107}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt - \hat{y}_{i,s} \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt \right) \tag{108}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \hat{y}_{i,s} \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt \right) \tag{109}$$

$$= \sum_{s=1}^{S} 2 \left(y_i \cdot \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \nabla_w \hat{y}_{w_s(0)}(x_i) \rangle dt - \int_0^1 \langle \nabla_w \hat{y}_{w_s(t))}(x), \frac{1}{2} \nabla_w (\hat{y}_{w_s(0)}(x_i))^2 \rangle dt \right) \tag{110}$$

$$\tag{111}$$

Otherwise, we get the usual

$$= \sum_{s=1}^{S} 2(y_i - \hat{y}_{i,s}) \int_0^1 \langle \nabla_w \hat{y}_{w_s(t,x))}(x), \nabla_w \hat{y}_{w_s(t,x)}(x') \rangle dt \tag{112}$$

$$\tag{113}$$

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 $\hat{y}(x_i)$.

A.4 Multi-Class Case

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

$$\text{Case 1: } L : \mathbb{R}^K \to \mathbb{R} \tag{114}$$

$$\text{Case 2: } L : \mathbb{R}^K \to \mathbb{R}^K \tag{115}$$

$$\tag{116}$$

A.4.1 Case 1 Scalar Loss

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

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

$$\frac{\partial \hat{y}}{\partial t} = \sum_{j=1}^{M} \frac{\partial \hat{y}(x)}{\partial w_j} \frac{\partial w_j}{\partial t} \tag{117}$$

$$= -\varepsilon \sum_{j=1}^{M} \frac{\partial \hat{y}(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(\hat{y}(x_i), y_i)}{\partial w_j} \tag{118}$$

$$\text{Apply chain rule} \tag{119}$$

$$= -\varepsilon \sum_{j=1}^{M} \frac{\partial \hat{y}(x)}{\partial w_j} \sum_{i=1}^{N} \frac{\partial L(\hat{y}(x_i), y_i)}{\partial \hat{y}} \frac{\partial \hat{y}(x_i)}{\partial w_j} \tag{120}$$

$$\text{Let} \tag{121}$$

$$A = \frac{\partial \hat{y}(x)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{122}$$

$$B = \frac{\partial L(\hat{y}(x_i), y_i)}{\partial \hat{y}} \in \mathbb{R}^{1 \times K} \tag{123}$$

$$C = \frac{\partial \hat{y}(x_i)}{\partial w_j} \in \mathbb{R}^{K \times 1} \tag{124}$$

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

$$(ABC)^T = (BC)^T A^T = BCA^T \tag{125}$$

$$ABC = (BCA^T)^T \tag{126}$$

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

the product as follows:

$$= -\varepsilon \sum_{j=1}^{M} \sum_{i=1}^{N} \left(\frac{\partial L(\hat{y}(x_i), y_i)}{\partial \hat{y}} \frac{\partial \hat{y}(x_i)}{\partial w_j} \left(\frac{\partial \hat{y}(x)}{\partial w_j} \right)^T \right)^T \tag{127}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{\partial L(\hat{y}(x_i), y_i)}{\partial \hat{y}} \sum_{j=1}^{M} \frac{\partial \hat{y}(x_i)}{\partial w_j} \left(\frac{\partial \hat{y}(x)}{\partial w_j} \right)^T \right)^T \tag{128}$$

$$\tag{129}$$

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

$$(\nabla \hat{y}(x))_{k,j} = \frac{\partial \hat{y}_k(x)}{\partial w_j} \tag{130}$$

$$= -\varepsilon \sum_{i=1}^{N} \left(\frac{\partial L(\hat{y}(x_i), y_i)}{\partial \hat{y}} \nabla \hat{y}(x_i) (\nabla \hat{y}(x))^T \right)^T \tag{131}$$

$$\tag{132}$$

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 $[K, 1]$

A.5 Multi Class Case

In the case where $\hat{y}$ is a vector we denote the data index by the superscript $y^{[i]}$ and the vector component by the subscript y_k.

query point may need to know about other class gradients than the target class.

k by k

yen ting's approach is needed because we are making some assumption about the model that allows us to measure the query change along a linear path.

The negative log likelyhood function where y and $\hat{y}$ are vectors.

$$NLL(y, \hat{y}) = \sum_{k}^{K} -y_k \ln(\hat{y}_k) \tag{133}$$

$$\frac{\partial \left[\hat{y}_0 ... y_k\right]}{dt} = \sum_{j=1}^{M} \frac{\partial \left[\hat{y}_0 ... \hat{y}_k\right]}{\partial w_j} \frac{\partial w_j}{dt} \tag{134}$$

$$= \sum_{j=1}^{M} \frac{\partial \left[\hat{y}_0 ... \hat{y}_k\right]}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{N} \frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial w_j} \right) \tag{135}$$

$$= \sum_{j=1}^{M} \frac{\partial \left[\hat{y}_0 ... \hat{y}_k\right]}{\partial w_j} \left(-\varepsilon \sum_{i=1}^{N} \frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial \left[\hat{y}_0^{[i]} ... \hat{y}_k^{[i]}\right]} \frac{\partial \left[\hat{y}_0^{[i]} ... \hat{y}_k^{[i]}\right]}{\partial w_j} \right) \tag{136}$$

$$= -\varepsilon \sum_{i=1}^{N} \frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial \left[\hat{y}_0^{[i]} ... \hat{y}_k^{[i]}\right]} \sum_{j=1}^{M} \frac{\partial \left[\hat{y}_0 ... \hat{y}_k\right]}{\partial w_j} \frac{\partial \left[\hat{y}_0^{[i]} ... \hat{y}_k^{[i]}\right]}{\partial w_j} \tag{137}$$

Now we use the fact that L is CCE

$$\frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial y_k^{[i]}} = \begin{cases} 0, k \neq y^{[i]} \\ -1 \end{cases} \tag{138}$$

$$\frac{\partial \hat{y}_k}{dt} = -\varepsilon \sum_{i=1}^{N} 1 \sum_{j=1}^{M} \frac{\partial \left[\hat{y}_0 ... \hat{y}_k\right]}{\partial w_j} \frac{\partial \left[\hat{y}_0^{[i]} ... \hat{y}_k^{[i]}\right]}{\partial w_j} \tag{139}$$

$$y_0 + \left[\frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial y_0^{[i]}} \sum_{j=1}^{M} \frac{\partial f(x_i)_0}{\partial w_j} \cdot \frac{\partial f(x)_0}{\partial w_j}, \frac{\partial L(\hat{y}^{[i]}, y^{[i]})}{\partial y_1^{[i]}} \sum_{j=1}^{M} \frac{\partial f(x_i)_1}{\partial w_j} \cdot \frac{\partial f(x)_1}{\partial w_j} \right] \tag{140}$$

B Neural Networks are Gaussian Processes

With Dropout on, the interpolant from one class to another will go into a variety of other classes. If you make a histogram of the locations where these boundary crossings occur, that will show a gaussian.

insert figure(s)

C prove path kernel result in context of differential flow of gradients on neural network. using forward euler approx of grad flow.

D ** look for sample in weight space and look for gradients

that are pointing toward the final point versus wanting a different direction. Then dot product those with the training direction.

E training gradients are smooth

F robust network types : regularized, michael's pca, Soft Nearest Neighbor Loss (SNNL) and adversarially trained.

G high dimensional arcs are very similar to chords

H linear interpolated model parameters from random to trained state yield robust models

H.1 For Mnist Inner Products in weight space matter more than distances

– does this generalize to ImNet?

I define robustness in terms of skew versus orthogonal

J define robustness in terms of attack perturbation magnitude

K Model Skewness and decision_boundaries.

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

References

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

[Carlini and Wagner(2016)] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. *arXiv:1608.04644 [cs]*, Aug. 2016. URL http://arxiv.org/abs/1608.04644. arXiv: 1608.04644.

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

[Goodfellow et al.(2013)Goodfellow, Bulatov, Ibarz, Arnoud, and Shet] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks, 2013.

[Goodfellow et al.(2014)Goodfellow, Shlens, and Szegedy] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *arXiv:1412.6572 [cs, stat]*, Dec. 2014. URL http://arxiv.org/abs/1412.6572. arXiv: 1412.6572.

[Hardt et al.(2015)Hardt, Recht, and Singer] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *CoRR*, abs/1509.01240, 2015. URL http://arxiv.org/abs/1509.01240.

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

[Kak(1993)] S. Kak. On training feedforward neural networks. *Pramana*, 40(1):35–42, 1993.

[Khoury and Hadfield-Menell(2018)] M. Khoury and D. Hadfield-Menell. On the geometry of adversarial examples. *CoRR*, abs/1811.00525, 2018. URL http://arxiv.org/abs/1811.00525.

[Krause(2020)] A. Krause. Introduction to machine learning, August 2020.

[Kurakin et al.(2016)Kurakin, Goodfellow, and Bengio] A. Kurakin, I. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *arXiv:1607.02533 [cs, stat]*, July 2016. URL http://arxiv.org/abs/1607.02533. arXiv: 1607.02533.

[LeCun et al.(1988)LeCun, Touresky, Hinton, and Sejnowski] Y. LeCun, D. Touresky, G. Hinton, and T. Sejnowski. A theoretical framework for back-propagation. In *Proceedings of the 1988 connectionist models summer school*, volume 1, pages 21–28. CMU, Pittsburgh, Pa: Morgan Kaufmann, 1988.

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

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

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

[Li and Yuan(2017)] Y. Li and Y. Yuan. Convergence analysis of two-layer neural networks with relu activation. In *Advances in Neural Information Processing Systems*, pages 597–607, 2017.

[Linnainmaa(1970)] S. Linnainmaa. 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, 1970.

[Liu and Deng(2015)] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian conference on pattern recognition (ACPR)*, pages 730–734. IEEE, 2015.

[Malik and Perona(1990)] J. Malik and P. Perona. Preattentive texture discrimination with early vision mechanisms. *JOSA A*, 7(5):923–932, 1990.

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

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

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

[Moosavi-Dezfooli et al.(2015)Moosavi-Dezfooli, Fawzi, and Frossard] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. DeepFool: a simple and accurate method to fool deep neural networks. *arXiv:1511.04599 [cs]*, Nov. 2015. URL http://arxiv.org/abs/1511.04599. arXiv: 1511.04599.

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

[Papernot et al.(2015)Papernot, McDaniel, Jha, Fredrikson, Celik, and Swami] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. *arXiv:1511.07528 [cs, stat]*, Nov. 2015. URL http://arxiv.org/abs/1511.07528. arXiv: 1511.07528.

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

[Prakash et al.(2018)Prakash, Moran, Garber, DiLillo, and Storer] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. A. Storer. Deflecting adversarial attacks with pixel deflection. *CoRR*, abs/1801.08926, 2018. URL http://arxiv.org/abs/1801.08926.

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

[Rumelhart et al.(1986)Rumelhart, Hinton, and Williams] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

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

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

[Szegedy et al.(2013)Szegedy, Zaremba, Sutskever, Bruna, Erhan, Goodfellow, and Fergus] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL http://arxiv.org/abs/1312.6199.

[Werbos(1974)] P. J. Werbos. 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, 1974.

[Wiyatno and Xu(2018)] R. Wiyatno and A. Xu. Maximal jacobian-based saliency map attack. *CoRR*, abs/1808.07945, 2018. URL http://arxiv.org/abs/1808.07945.