# DEEP HYPERSPHERICAL DEFENSE AGAINST ADVERSARIAL PERTURBATIONS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Recent studies show that deep neural networks are extremely vulnerable to adversarial examples which are semantically indistinguishable from natural data and yet incorrectly classified. These adversarial examples are generated from the natural data by adding a small amount of adversarial perturbation. This paper tackles the adversarial attack problem with hyperspherical defense - a defense strategy that learns neural network over hyperspheres. The hyperspherical defense framework is well motivated by: (i) Learning on hyperspheres gives us bounded output, which may make the geometry of neural networks more smooth; (ii) Learning on hyperspheres could naturally eliminate some adversarial perturbations and reduce the effect of adversarial perturbations; (iii) Representing data on hyperspheres selectively drops some information of the inputs, but these information are shown to be not crucial to visual recognition (based on the fact that hyperspherical neural network performs comparable to or even better than standard neural networks in visual recognition). Furthermore, we introduce the hyperspherical compactness and propose a robust geodesic inference. We also provide theoretical insights about why our hyperspherical defense improves robustness. Last, we validate the superiority of hyperspherical defense with extensive experiments on both white-box and black-box adversarial attacks on multiple datasets.

## 1 INTRODUCTION

Recent years have witnessed great breakthroughs in deep learning in a variety of applications ranging from face recognition (Taigman et al., 2014), object detection/recognition (Girshick et al., 2014; He et al., 2016) and speech recognition (Amodei et al., 2016) to machine translation (Sutskever et al., 2014) and gaming (Silver et al., 2016). However, recent studies show that most of these machine learning models (especially deep models) are very vulnerable to *adversarial attacks*. Such attacks can seriously undermine the security of a system supported by the deep learning models, causing the reliability problem in autonomous driving, biometric authentication, etc. Specifically, attackers transform a normal sample to a malicious one by simply adding a small amount of perturbations. These malicious samples (i.e. *adversarial examples*) are semantically indistinguishable from the normal ones, but usually will be incorrectly classified by deep learning models.

Due to the significance in practice, both adversarial attack and defense in deep learning have received particular attention. While we have seen many work on generating successful attacks (Szegedy et al., 2013; Goodfellow et al., 2014; Nguyen et al., 2015), finding an effective defense strategy remains an open challenge. In this paper, we present a simple yet effective hyperspherical defense framework to improve the robustness against adversarial attacks. The basic idea is to constrain the network learning onto a hypersphere. Specifically for CNNs, we basically replace the inner product between the kernel and the local patch with *hyperspherical correlation*. The hyperspherical correlation is defined as a function of the angle between two input vectors (feel free to think about $\cos(\theta)$ as a simple example). The hyperspherical correlation is supposed to be upper and lower bounded in general, making the output of each layer also bounded. Because hyperspheres are the simplest Riemannian manifold, we are essentially transforming the learning space from Euclidean space to a manifold space. Our direct intuition comes from the observation from Fig. 1. As one could see from Fig. 1, the geodesic (angular) distance on the unit sphere corresponds to the semantic difference, while the distance on the radius direction corresponds to the intra-class variation (their labels are the same). Because the adversarial examples usually will not change the semantic meaning of an

image, it is very likely that the adversarial perturbation has large components on the radius direction (verified by Fig. 2(a)). Naturally, it inspires us to learn the neural network on the hypersphere such that many adversarial perturbations could alleviated or eliminated. In the following sections, we will give more empirical and theoretical justifications for this observations.

Our motivations to learn a neural network on hyperspheres against adversarial attacks mainly lie in three aspects. First, learning on unit hyperspheres can increase the smoothness of the network (more smooth than the standard neural networks). As one may see, many of the recent works (Cisse et al., 2017; Miyato et al., 2015) highly favor the smoothness in the network, because it can make the effect of adversarial perturbations small and also empirically perform well. They usually achieve the smoothness by modifying learning objectives or adding additional constraints, while ours achieves it in a more explicit way by directly constraining the network to learn on a unit hypersphere. Second, projection from Euclidean space onto hyperspheres naturally alleviates the effect of some adversarial perturbations. For example, if the adversarial perturbations point to the radius direction of a hyphersphere, then such perturbations will not affect the final prediction. Third, we observe that hyperspherical neural networks achieve comparable or better performance in visual recognition with much smaller parameter space (Liu et al., 2017), indicating that it preserves the semantic information (information related to the label) of an image. As we know, the adversarial perturbations will not affect the semantic meaning of an image, so it is very likely that learning on hyperspheres can drop most of the components of the adversarial perturbations.

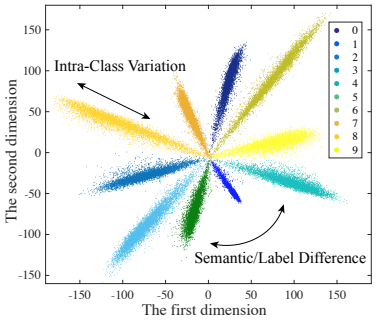



Figure 1: 2D CNN Features on MNIST.

Our major contributions can be summarized as follows:

- We introduce a simple yet effective robust strategy to train a neural network with strong robustness against adversarial perturbations. Specifically, we present a defensive hyperspherical neural networks (D-SphereNet) that are learned on hyperspheres, and propose two new hyperspherical correlations: quadratic one and input-truncated one.
- Besides adapting hyperspherial learning to train a robust neural network, we further propose several learning objectives towards hyperspherical compactness and robust inference methods.
- We also give partial theoretical insights for the advantages of utilizing hyperspherical defense to improve robustness against adversarial attacks.
- Most importantly, our D-SphereNet yields strong and consistent empirical performance in resisting both *black-box* and *white-box* adversarial attacks.

## 2 Related work

There is an increasingly growing body of work on generating adversarial examples (Szegedy et al., 2013; Goodfellow et al., 2014; Nguyen et al., 2015). Szegedy et al. (2013) observes that a deep convolutional neural networks (CNNs) can easily fail with a small amount of artificial perturbations. They use box-constrained Limited-memory BFGS algorithm to obtain adversarial examples that are transferrable across different neural network architectures. Goodfellow et al. (2014) make an attempt to explain that it is the linear nature of CNNs that causes such vulnerabilities. They also propose the fast gradient sign method (FGSM) to efficiently generate adversarial examples. Some recent work (Papernot et al., 2016b; Fawzi et al., 2015; Nguyen et al., 2015; Moosavi-Dezfooli et al., 2016b;a; Madry et al., 2017) also explore adversarial attacks in depth and try to find other ways to generate adversarial examples accurately and efficiently. These adversarial examples are generally transferable across different network architectures, which also makes the black-box attacks possible.

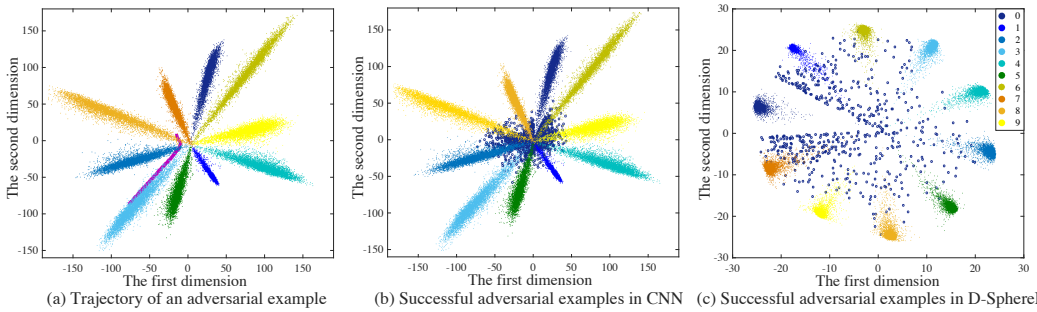
Current most prevailing defense strategies can be viewed as modifying objective functions, such as adversarial training (Szegedy et al., 2013; Goodfellow et al., 2014), smooth regularization (Miyato et al., 2015; Zheng et al., 2016), defensive distillation (Papernot et al., 2016c), Parseval training

(Cisse et al., 2017) etc. One of the most effective defense by far is the adversarial training (Szegedy et al., 2013; Goodfellow et al., 2014). It incorporates adversarial examples into the training stage to improve the robustness, so it is time-consuming and may only be effective to the adversarial examples that have appeared in the training stage. Lu et al. (2017) construct a detection network to detect and reject the adversarial examples, but it does not improve the robustness of deep models. Different from the current defense strategies, hyperspherical defense effectively combines robust learning architecture, compactness learning objective and improved inference method into a unified framework and significantly improves the robustness of deep models.

Our hyperspherical defense is largely inspired by hyperspherical learning which is first proposed in (Liu et al., 2017) mostly as a way to improve the convergence of learning neural networks. Different from (Liu et al., 2017), our hyperspherical defense framework generalizes the the hyperspherical convolutional operator to a more general concept - hyperspherical correlation (SphereCorr). Besides generalizing the exisitng hyperspherical convolutions to SphereCorr, we also propose the quadratic SphereCorr to introduce more non-linearity and the input-truncated SphereCorr for stronger robustness. Furthermore, we utilize a hyperspherical compactness objective function and a robust inference method, which can effectively resist and reject adversarial examples. We also generalize the hyperspherical setup to a mixed setup of hypersphere and simplices. This not only makes learning more flexible, but also make it difficult when one hacks the structure of the neural network (black-box attack).

## 3 Some Observations about Adversarial Perturbations

Before delving into the details, we first motivate the readers with a few empirical observations about adversarial perturbations. Our initial intuitions and motivations are built upon them.




(a) Trajectory of an adversarial example (b) Successful adversarial examples in CNN (c) Successful adversarial examples in D-SphereNet

Figure 2: 2D CNN Features on MNIST. These 2D features are directly learned from the neural network by setting the feature dimension as 2. Feature dimension refers to the output dimension of the fully connected layer before the 10-class classifiers. These 2D features are true distributions from the neural network, rather than generated some visualization methods. Fig. 1 is also plotted using the same method.

**Adversarial examples tend to approach the origin first and then shift to another class.** In Fig. 2(a), we draw the trajectory (purple line) of an adversarial example. The trajectory is computed by varying $\lambda$ from 0 to 1 in $\boldsymbol{x}+\lambda\cdot\text{FGSM}$ where $\boldsymbol{x}$ is the normal sample and FGSM denotes the adversarial perturbations generated by the FGSM. Specifically, we use a normal image from digit 3 and gradually increase the magnitude of the adversarial perturbation. We observe from Fig. 2(a) that the adversarial example will approach to the origin first and then start shifting to another class. The trajectory indicates that it is easier to maliciously change the predicted label of a network from a less semantic space[1]. It motivates us to learn representation on a hypersphere, which can cause the adversarial perturbation much more effort in order to fool the network (the adversarial perturbations can no longer approach to the origin to affect the output label distribution).

**Adversarial examples is more seperable from the normal data in D-SphereNet.** In Fig. 2(b) and Fig. 2(c), we plot the adversarial examples of digit 0 that have successfully fooled the network. Fig. 2(b) shows that the adversarial examples of standard CNNs overlap with a large number of natural data. In contrast, we can see in Fig. 2(c) that the adversarial examples of D-SphereNets

[1]We view the feature space near the origin as a less semantic space, because in origin it only requires small perturbations to change from one class to another.

are clearly separable from all the natural data, which also means the adversarial examples will not appear in where the natural data are dense. It clearly shows the advantages of D-SphereNets.

**Hyperspherical defense can better group normal data from different classes.** We could also observe from Fig. 2(b) and Fig. 2(c) that the D-SphereNets could better group the natural data of the same class together. The embedding space[2] for natural data in D-SphereNets are much more compact than standard CNNs, which motivates us to propose the robust inference.

# 4 HYPERSPHERICAL DEFENSE

## 4.1 DEFENSIVE HYPERSPHERICAL NEURAL NETWORKS

The proposed D-SphereNet can be viewed as a neural network that learns its parameters on hyperspheres. To achieve this, we measure the correlation between the input and weights on hyperspheres instead of the original inner product. The hyperspherical correlation (SphereCorr) is defined as

$$\langle \boldsymbol{w}, \boldsymbol{x} \rangle_{\mathcal{M}} = g(\boldsymbol{w}, \boldsymbol{x}) \tag{1}$$

which can be viewed as a projected inner product defined on hypersphere $\mathcal{M}$. Similar to (Liu et al., 2017), $g(\boldsymbol{w}, \boldsymbol{x})$ have different instantiations.

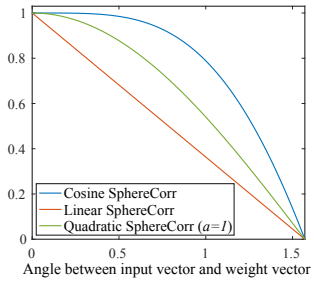



Figure 3: Different SphereCorr.

**Cosine SphereCorr.** The simplest form is the cosine hyperspherical correlation: $g(\boldsymbol{w}, \boldsymbol{x}) = \cos(\theta_{\boldsymbol{w},\boldsymbol{x}})$ where $\theta_{\boldsymbol{w},\boldsymbol{x}}$ denotes the angle between the input vector $\boldsymbol{x}$ and the weight vector $\boldsymbol{w}$. The cosine SphereCorr is closely related to the geodesic distance on the unit hypersphere. Following (Liu et al., 2017), we could also derive linear SphereCorr $g(\boldsymbol{w}, \boldsymbol{x}) = a\theta_{\boldsymbol{w},\boldsymbol{x}} + b$, sigmoid SphereCorr, etc. Generally, we make the output of SphereConv to be bounded from below and above and decrease monotonically with the angle $\theta_{\boldsymbol{w},\boldsymbol{x}}$.

**Quadratic SphereCorr.** Inspired by the observation that the RBF networks have nearly no adversarial examples and the claim that adversarial examples are partially caused by the neural network being too "linear" (Goodfellow et al., 2014), we increase the non-linearity by incorporating a quadratic effect to form a composite function:

$$g(\boldsymbol{w}, \boldsymbol{x}) = 1 - a(1 - \cos(\theta_{\boldsymbol{w},\boldsymbol{x}}))^2 \tag{2}$$

where $a > 0$ is a parameter that controls the curvature of the quadratic SphereCorr. In this paper, we mainly consider the case where $a = 1$ to make the output the same as the other SphereCorr. The quadratic SphereCorr can bring the quadratic non-linearity to the neural network and potentially make the network more non-linear. From Fig. 3, we could have an intuitive sense for these SphereCorr functions. Linear SphereCorr and Cosine SphereCorr are two extreme cases, while the quadratic SphereCorr is somewhere in the middle. Because we usually use rectified linear units (ReLUs) in the network, we only show the valid range $[0, \frac{\pi}{2}]$ instead of $[0, \pi]$ in Fig. 3.

**Input-Truncated SphereCorr**. If the norm of the input $\|\boldsymbol{x}\|$ is larger than 1, the input-truncated SphereCorr will behave exactly like cosine SphereCorr. Differently, if $\|\boldsymbol{x}\|$ is smaller than 1, the input-truncated SphereCorr will preserve $\|\boldsymbol{x}\|$ instead of normalizing it to 1. The specific form is

$$g(\boldsymbol{w}, \boldsymbol{x}) = \min(1, \|x\|) \cos(\theta_{\boldsymbol{w},\boldsymbol{x}}) \tag{3}$$

This SphereConv is still bounded in $[-1, 1]$ and is especially robust in the sense that it could better reduce the perturbation inside the hypersphere. For example, given $\|\boldsymbol{x}\| < 1$, then its direction will largely affect the output of the other SphereCorrs. However, because we have $\|\boldsymbol{x}\|$ in $g(\boldsymbol{w}, \boldsymbol{x})$, the multiplicative factor $\|\boldsymbol{x}\| < 1$ can reduce the effect of perturbations within the hypersphere. Thus the input-truncated SphereCorr can adaptively reduce adversarial perturbations and achieve robustness.

**Optimization.** Because we replace the original inner product similarity with the SphereCorr, it will cause some minor differences in optimization. We still use the back-prop to learn the parameters of the network. Because $\theta_{\boldsymbol{w},\boldsymbol{x}}$ is an intermediate variable, we need to use the chain rule to compute the

[2]The embedding space is the feature space before entering the classifier (*i.e.*, last fully connected layer).

gradient with respect to $\boldsymbol{w}$ and $\boldsymbol{x}$. We have $\frac{\partial g(\boldsymbol{w},\boldsymbol{x})}{\partial \boldsymbol{w}} = \frac{\partial g(\boldsymbol{w},\boldsymbol{x})}{\partial \theta_{\boldsymbol{w},\boldsymbol{x}}} \cdot \frac{\partial \theta_{\boldsymbol{w},\boldsymbol{x}}}{\partial \boldsymbol{w}}$ and $\frac{\partial g(\boldsymbol{w},\boldsymbol{x})}{\partial \boldsymbol{x}} = \frac{\partial g(\boldsymbol{w},\boldsymbol{x})}{\partial \theta_{\boldsymbol{w},\boldsymbol{x}}} \cdot \frac{\partial \theta_{\boldsymbol{w},\boldsymbol{x}}}{\partial \boldsymbol{x}}$ where $\theta_{\boldsymbol{w},\boldsymbol{x}} = \cos(\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|\|\boldsymbol{x}\|})$. With these chain rule computation, it is trivial to perform back-prop.

**Learning objective.** Because we are representing features on Hyperspheres and all the similarities are also measured on hyperspheres, we also need to adapt the original softmax loss[3] to our hyperspherical learning. Considering the binary classification, we have that the original decision boundary is $\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_{\boldsymbol{W}_1,\boldsymbol{x}}) = \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_{\boldsymbol{W}_2,\boldsymbol{x}})$ where $\boldsymbol{W}_i$ is the classifier for the $i$-th class and $\boldsymbol{x}$ is the feature outputted from the fully connected layer before classifiers. To make the decision merely depend on geodesic distance on unit hyperspheres, we only need to normalize the $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ to the same length (we use length 1 here). Therefore, we usually use the weight-normalized softmax (W-Softmax) loss in D-SphereNet instead of original softmax loss.

**Encouraging Orthogonality.** While we are computing all the similarities based on the unit hypersphere, we will no longer use the weight decay in our network. To encourage the direction of weights to be uniformly spaced out, we impose add an orthogonality-inducing regularizer. Specifically, given a set of kernels $\boldsymbol{W}$ where the $i$-th column vector $\boldsymbol{W}_i$ is the weights of the $i$-th kernel, the orthogonality-inducing regularizer is written as $\|\boldsymbol{W}^\top\boldsymbol{W} - \boldsymbol{I}\|_F^2$, where $\boldsymbol{I}$ is identity matrix.[4]

**Convolutional layers and fully connected layers in D-SphereNets.** Using SphereCorr, we could easily transform the standard convolutional layers and fully connected layers into the ones in D-SphereNet by replacing the inner product with the SphereCorr.

### 4.2 Hyperspherical Compactness

From Fig. 2(b) and Fig. 2(c), we notice that if a testing sample is not close to any training data clouds, then it is very likely to be an adversarial example. In order to better distinguish the adversarial examples and the normal data, we need to make the features in the embedding space more compact and well grouped. To this end, we propose some learning objectives for hyperspherical compactness in this section. We start by looking into the W-Softmax loss:

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{\|\boldsymbol{x}_i\| g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)}}{\sum_j e^{\|\boldsymbol{x}_i\| g(\boldsymbol{W}_j,\boldsymbol{x}_i)}}\right) \tag{4}$$

where $g(\cdot)$ is the SphereCorr, $y_i$ is the label of the sample $\boldsymbol{x}_i$, $\boldsymbol{W}_j$ is the classifier corresponding to the $j$-th class (*i.e.*, the weights of the last fully connected layer). There are two ways to minimize the W-Softmax loss to a low value. One is to increase $g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)$, while the other way is to increase $\|\boldsymbol{x}_i\|$ given that $g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)$ is already larger than $g(\boldsymbol{W}_j,\boldsymbol{x}_j), j \neq y_i$. As a result, we can not guarantee that the learned features have strong hyperspherical compactness. In order to guarantee these features to be well grouped on hyperspheres, the simplest way is to constrain the norm of the column weights to a small value. However, if such value is too small, the optimization is too difficult and the loss will not decrease at all. To address this, we design two new objective functions:

**Weight multiplier.** Following the idea of constraining the norm of weights, we use $\alpha$ to denote $\max_j \|\boldsymbol{W}_j\|$. Then we can instead optimize the following constrained objective function:

$$\underset{\boldsymbol{W},\mathrm{DSN}(\boldsymbol{x}_i)}{\operatorname{argmin}} \frac{1}{N}\sum_i -\log\left(\frac{e^{\alpha\cdot g(\boldsymbol{W}_{y_i},\mathrm{DSN}(\boldsymbol{x}_i))}}{\sum_j e^{\alpha\cdot g(\boldsymbol{W}_j,\mathrm{SN}(\boldsymbol{x}_i))}}\right) \quad \text{s.t. } 0<\alpha\leq m \tag{5}$$

where $m$ is a preset parameter and $\mathrm{DSN}(\boldsymbol{x}_i)$ is the weights of the D-SphereNet with the input sample $\boldsymbol{x}_i$. Because the inequality constraint is difficult to handle in neural network, we approximate it with a unconstrained optimization which the neural network is easy to deal with:

$$\underset{\boldsymbol{W},\mathrm{DSN}(\boldsymbol{x}_i)}{\operatorname{argmin}} \frac{1}{N}\sum_i -\log\left(\frac{e^{|\alpha|\cdot g(\boldsymbol{W}_{y_i},\mathrm{DSN}(\boldsymbol{x}_i))}}{\sum_j e^{|\alpha|\cdot g(\boldsymbol{W}_j,\mathrm{SN}(\boldsymbol{x}_i))}}\right) + \eta\cdot|\alpha| \tag{6}$$

where $\eta$ is a hyperparameter that controls how small we want the norm of column weights to be.

[3]The softmax loss is defined as the combination of the last fully connected layer (*i.e.*, classifiers), softmax function and the cross-entropy loss.

[4]We do not directly apply an orthogonal transformation to $W$, because we find such a regularizer is more computationally efficient in practice. We also adopt the stochastic manifold gradient optimization, since the orthogonality constraint is essentially the Stefiel Manifold.

**Loss annealing.** Alternatively, we could introduce a sphere-normalized softmax loss defined as:

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)}}{\sum_j e^{g(\boldsymbol{W}_j,\boldsymbol{x}_i)}}\right) \tag{7}$$

which simultaneously normalize both the column weighs $\boldsymbol{W}_j$ and the input feature $\boldsymbol{x}_i$. However, this loss function defines a very difficult task that can not even converge. So we use an annealing method to optimize such loss function. Eventually, we optimize a loss function as follows:

$$L = -\frac{1}{N}\sum_i \frac{1}{1+\lambda}\log\left(\frac{e^{g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)}}{\sum_j e^{g(\boldsymbol{W}_j,\boldsymbol{x}_i)}}\right) + \frac{\lambda}{1+\lambda}\log\left(\frac{e^{\|\boldsymbol{x}_i\| g(\boldsymbol{W}_{y_i},\boldsymbol{x}_i)}}{\sum_j e^{\|\boldsymbol{x}_i\| g(\boldsymbol{W}_j,\boldsymbol{x}_i)}}\right) \tag{8}$$

where $\lambda$ starts with a relatively large value and is usually a monotonically decreasing function (in terms of iteration number) with a non-negative minimum. The intuition hehind is to learn the network from a easy task (loss) first and gradually shift to a difficult task (loss).

### 4.3 Robust Geodesic Inference

To take full advantage of the hyperspherical compactness, we further design a robust geodesic inference method that could help to reject adversarial examples and achieve robust recognition. The geodesic inference is a simple two-step approach. We will first calculate the mean $\theta_{\mu_j}$ and the standard deviation $\theta_{\sigma_j}$ of angles between all the $j$-th class training samples and classifier $\boldsymbol{W}_j$ (*i.e.*, the $j$-th column weights of the last fully connected layer).

**Step 1.** Given a testing sample, we use the softmax probability to perform standard neural network inference and output the prediction label $\widehat{y}_{\text{te}}$. We can also obtain the features $\boldsymbol{x}_{\text{te}}$ before entering the last fully connected layer.

**Step 2.** We compute the angle $\widehat{\theta}$ between $\boldsymbol{x}_{\text{te}}$ and the classifier $\boldsymbol{W}_{y_{\text{te}}}$. If the angle $\widehat{\theta}$ is larger than the angular threshold $p(\theta_{\mu_j} + \theta_{\sigma_j})$ where $p$ is a hyperparameter controlling this threshold, we reject this testing sample and regard it as adversarial attack. If $\widehat{\theta}$ is smaller than the threshold, we accept it and the final prediction is its output label in Step 1. Note that, we usually set $p$ to 1, but it could also be set using cross validation.

### 4.4 Generalized Hyperspherical Defense

For now, we are still working on the $\ell_2$ hypersphere which is one of the simplest manifold. Could we actually generalize to different manifolds? In fact, we could easily generalize our hyperspherical defense framework to generalized hyperspheres (e.g., simplex, hyperbola, affine). This not only make the architecture more flexible, but also make it more difficult if one wants to hack the architecture of the neural networks.

## 5 Theoretical Insights

Before we proceed, we denote some notations for simplicity.

**Notations**: Given a vector $\boldsymbol{x} = (x_1, ..., x_p)^\top \in \mathbb{R}^p$, $\text{sign}(\boldsymbol{x}) := (\text{sign}(x_1), ..., \text{sign}(x_p))^\top$, we define vector norms: $\|\boldsymbol{x}\|_1 = \sum_j |x_j|$, $\|\boldsymbol{x}\|_2^2 = \sum_j (x_j)^2$. Define $\mathcal{S}^{p-1} = \{\boldsymbol{x} \in \mathbb{R}^p : \|\boldsymbol{x}\|_2 = 1\}$.

We provide theoretical insights of the hyperspherical defense in both adversarial training and attack. Here is the high level intuition: The black-box adversarial attack assumes that the inputs of each layer lie in the Euclidean space, and perturbs them in the Euclidean space. However, the inputs of each layer in the D-Spherenet lie on a spherical manifold, and the adversarial perturbation can be naturally reduced by "projecting the input onto the sphere" in each layer.

Due to the complex structures of the D-Spherenet, we illustrate the effectiveness of the projection using an example based on linear model, which can be viewed as a single neuron in the network. Similar approaches have been used in justifying other deep learning techniques such as dropout (Srivastava et al., 2014; Wager et al., 2013). Before we proceed, we first introduce the following linear model.

**Assumption 1.** *We consider a linear model with random design,*

$$y = \boldsymbol{x}^\top \boldsymbol{w}^* + \epsilon,$$

*where feature vector $\boldsymbol{x} \in \mathbb{R}^p$ is uniformly distributed over $\mathcal{S}^{p-1}$, i.e., $\boldsymbol{x} \sim \mathrm{Unif}(\mathcal{S}^{p-1})$, $\epsilon \sim N(0, \sigma^2)$ is the observation noise and independent on $\boldsymbol{x}$, and $\boldsymbol{w}^*$ is a dense regression coefficient vector.*

Here we assume that $\boldsymbol{w}^*$ is dense, because the D-SphereNet is not learning a sparse neural network. Moreover, $\sigma$ is also assumed to be small such that the signal-noise-ratio is sufficient large for learning.

Given the linear model defined in Assumption 1, we further explicitly characterize the FGSM attack in the following proposition.

**Proposition 1.** *Suppose Assumption 1 holds, we train the model using the square loss function*

$$\ell(\boldsymbol{x}, \boldsymbol{w}) = \frac{1}{2}(y - \boldsymbol{x}^\top \boldsymbol{w})^2.$$

*Given a sample $(\boldsymbol{x}, y)$, FSGM essentially generates an adversarial sample $(\widetilde{\boldsymbol{x}}, y)$ by*

$$\widetilde{\boldsymbol{x}} := \boldsymbol{x} - \delta \operatorname{sign}(\nabla_{\boldsymbol{x}} \ell(\boldsymbol{x}, \boldsymbol{w})) = \boldsymbol{x} - \delta \cdot \operatorname{sign}(\epsilon) \cdot \operatorname{sign}(\boldsymbol{w}^*). \tag{9}$$

Here $\delta$ is assumed to be very small, since the FGSM needs to generate semantically indistinguishable adversarial samples. As can be seen from Proposition 1, the FGSM perturbation is proportional to $\operatorname{sign}(\boldsymbol{w}^*)$ up to sign change (determined by $\epsilon$ associated with $\boldsymbol{x}$).

### 5.1 Adversarial Training

The next theorem shows that training the model using adversarial sample is essentially adding bias to estimation.

**Theorem 2.** *Suppose Assumption 1 holds, we train the model using the adversarial training sample generated by FGSM. We solve:*

$$\widehat{\boldsymbol{w}}_1 := \operatorname*{argmin}_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{x},\epsilon} \left[ (y - \widetilde{\boldsymbol{x}}^\top \boldsymbol{w})^2 \right]$$

*Given sufficiently small $\delta$ and $\sigma$, we have the following approximation:*

$$\widehat{\boldsymbol{w}}_1 \approx \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{p} \|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 + \left( \delta \|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}} \sigma \right)^2. \tag{10}$$

The proof of Theorem 2 is provided in the Appendix B.1. Theorem 2 implies that training model using adversarial samples is essentially solving an $\ell_1$-penalized least square problem. However, since the parameter of our interest $\boldsymbol{w}^*$ is a dense vector (as in Assumption 1), adding such an $\ell_1$ norm penalty induces estimation bias. Thus, it does not help, but harms the training. Note that our theoretical analysis only considers a linear model, which can be viewed as a single neuron. For a deep neural network, there are multiple layers with many neurons. Thus, the estimation bias at each neuron is passed to the next layer. When these biases are eventually accumulated at the last layer, the neural network can be significantly fooled.

The next theorem shows that by projecting $\widetilde{\boldsymbol{x}}$ to the sphere, we can obtain a estimate.

**Theorem 3.** *Suppose Assumption 1 holds, we train the model using the adversarial samples projected to spheres. We solve:*

$$\widehat{\boldsymbol{w}}_2 := \operatorname*{argmin}_{\boldsymbol{w}} \mathbb{E}_{\boldsymbol{x},\epsilon} \left[ \left( y - \frac{1}{\|\widetilde{\boldsymbol{x}}\|_2} \widetilde{\boldsymbol{x}}^\top \boldsymbol{w} \right)^2 \right].$$

*Given sufficiently small $\delta$ and $\sigma$, then we have the following approximation:*

$$\widehat{\boldsymbol{w}}_2 \approx \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{p} \|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 + \frac{p-2}{p} \Big( \delta \|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}} \sigma \Big)^2 + \delta^2 \Big( \frac{1}{p+2} \|\boldsymbol{w}\|_2^2 + \frac{2}{p(p+2)} \|\boldsymbol{w}\|_1^2 \Big). \tag{11}$$

The proof of Theorem 3 is provided in the Appendix B.2. Theorem 3 indicates that training the model with the adversarial samples projected onto the spheres improves the estimation. Specifically, we refine (11) in the following corollary.

**Corollary 4.** *Suppose Assumption 1 holds, given* $\widehat{\boldsymbol{w}}_1$ *in* (10) *and* $\widehat{\boldsymbol{w}}_2$ *in* (11)*, then*

$$\|\widehat{\boldsymbol{w}}_1 - \boldsymbol{w}^*\|_2 \geq \|\widehat{\boldsymbol{w}}_2 - \boldsymbol{w}^*\|_2 . \tag{12}$$

*Proof.* Note for any $\boldsymbol{w} \in \mathbb{R}^p$, we have

$$\begin{aligned} &\frac{p-2}{p}\big(\delta\|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}}\sigma\big)^2 + \delta^2\big(\frac{1}{p+2}\|\boldsymbol{w}\|_2^2 + \frac{2}{p(p+2)}\|\boldsymbol{w}\|_1^2\big) \\ \leq &\frac{p-2}{p}\big(\delta\|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}}\sigma\big)^2 + \frac{\delta^2}{p}\|\boldsymbol{w}\|_1 \leq \frac{p-1}{p}\big(\delta\|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}}\sigma\big)^2. \end{aligned} \tag{13}$$

The second inequality holds for $\|\boldsymbol{w}\|_2^2 \leq \|\boldsymbol{w}\|_1^2$. We view (10) and (11) as two Lagrangian formula. Notice there exists an one-to-one map between the Lagrangian formula and the constrained forms of the optimization problems. We have

$$\widehat{\boldsymbol{w}}_1 = \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{p}\|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 \ \text{ s.t. } \boldsymbol{w} \in C_1. \quad \text{and } \widehat{\boldsymbol{w}}_2 \quad = \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{p}\|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 \ \text{ s.t. } \boldsymbol{w} \in C_2. \tag{14}$$

Essentially, (13) reveals that $C_1 \subset C_2$. Then $\widehat{\boldsymbol{w}}_1$ is just one feasible solution in $C_2$. Therefore, by the optimal property of $\widehat{\boldsymbol{w}}_2$, we have the desired result. □

Corollary 4 shows that by projecting the adversarial samples to the spheres, we alleviate the regularization, and therefore reduce the estimation bias. Note that our theoretical analysis only consider a linear model, which can be viewed as a single neuron. For a deep neural network, there are multiple layers with many neurons, and we apply such a projection step at every neuron. Therefore, the overall bias (attack) reduction at these neurons is significant. This justifies the superiority of the D-SphereNet in adversarial training.

### 5.2 Robustness Against Adversarial Attack

We then provide theoretical insights of the hyperspherical defense when predicting adversarial testing samples.

**Theorem 5.** *Suppose Assumption 1 holds. Given a new input* $\widehat{\boldsymbol{x}} \sim$ *Unif*$(\mathcal{S}^{p-1})$ *with an associated noise* $\epsilon$ *and the adversarial example generated by* (9)*, we have*

$$\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|1/\|\widetilde{\boldsymbol{x}}\|_2 \cdot \widetilde{\boldsymbol{x}}^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2 \leq \frac{p-1}{p}\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\widetilde{\boldsymbol{x}}^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2. \tag{15}$$

The proof of Theorem 5 is provided in the Appendix B.3. Theorem 5 suggests by projecting the adversarial testing sample to the sphere, we can also improve prediction. Note that our theoretical analysis only consider a linear model, which can be viewed as a single neuron. For a deep neural network, there are multiple layers with many neurons. Thus, the improvement at each neuron is passed to the next layer. When these improvements are eventually accumulated at the last layer, we obtain significant better prediction. This reveals why the D-SphereNet is more robust against adversarial attack than other deep networks.

## 6 Discussions

**Why hyperspherical defense is useful in practice.** While facing white-box attacks, hyperspherical defense is very likely to reduce or even eliminate the effect of adversarial examples. While facing black-box attacks, hyperspherical defense has its unique advantages. If the attackers assume we are using standard CNNs, they can not get good attack performance since we are using D-SphereNets. If the attackers know we are using D-SphereNets, they will even get worse results if they do not have access to our exact network parameters (see black-box attack experiments). In practice, one could also randomize over different manifold setup to prevent attackers to explore the manifold structure.

**Comparisons and connections.** Our hyperspherical defense is designed to improve robustness against adversarial perturbations. Compared to (Liu et al., 2017), the hyperspherical defense framework has different robust SphereCorrs, new training objectives towards hyperspherical compactness, robust geodesic inference, and aims for totally different task. The only thing in common is that both (Liu et al., 2017) and hyperspherical defense use the hypersphere concepts. Similar to (Miyato et al., 2015; Zheng et al., 2016; Cisse et al., 2017), our framework also improves smoothness in neural networks, but we achieve this in a completely different way. We effectively combine novel architectures, compactness training, robust inference into a unified framework.

Table 1: White-box attack on MNIST dataset.

| Attack | Baseline CNN | | | D-SphereNet | | | D-SphereNet + WM | | |
|---|---|---|---|---|---|---|---|---|---|
| | Natural Train | FGSM Train | PGD Train | Natural Train | FGSM Train | PGD Train | Natural Train | FGSM Train | PGD Train |
| Natural Data | 99.36 | 99.43 | 99.29 | 99.50 | 99.51 | 99.40 | 99.37 | 99.37 | 99.44 |
| FGSM | 29.42 | 98.76 | 98.04 | 84.70 | 98.81 | 98.36 | 82.98 | 82.81 | 98.23 |
| PGD | 00.51 | 01.45 | 94.18 | 02.31 | 10.16 | 95.96 | 12.19 | 35.41 | 94.58 |

| Attack | D-SphereNet + WM + Quad SphereCorr | | | D-SphereNet + Loss Annealing | | | D-SphereNet + WM + Input-truncated Corr | | |
|---|---|---|---|---|---|---|---|---|---|
| | Natural Train | FGSM Train | PGD Train | Natural Train | FGSM Train | PGD Train | Natural Train | FGSM Train | PGD Train |
| Natural Data | 99.48 | 99.34 | 99.36 | 99.22 | 99.47 | 99.49 | 99.63 | 99.59 | 99.44 |
| FGSM | 89.62 | 98.51 | 95.82 | 88.39 | 98.22 | 98.18 | 76.52 | 99.06 | 98.09 |
| PGD | 14.84 | 26.34 | 82.81 | 17.12 | 34.67 | 94.66 | 03.48 | 19.10 | 95.02 |

# 7 EXPERIMENTS

## 7.1 EXPERIMENTAL SETTINGS

**General.** We put the detailed network architecture in the appendix. WM denotes the weight multiplier in all the tables. Our CNN models are trained with ADAM with initial learning rate as $0.001$. In MNIST, we divide the learning rate by 10 every 10K iterations and stop at 30K iterations. In CIFAR-10, we divide the learning rate by 10 at 34K and 54K iterations and stop at 64K iterations. For all experiments, we use the CleverHan library (Papernot et al., 2016a).

**White-box Attack.** For MNIST, we train both baseline CNN and D-SphereNet model with natural, FGSM and PGD training. Specifically, for FGSM training, we set $\epsilon = 0.3$. For PGD training we set $\epsilon = 0.3$, step size as $0.75$ and number of iterations as $10$. We evaluate the D-SphereNets with different SphereCorrs and different learning objectives. For CIFAR-10, we train both vanilla and D-SphereNet models, including variants models, with natural and FGSM training. Specifically, for FGSM training, we set $\epsilon = 8/255$ and for PGD training we set $\epsilon = 8/255$, step size as $2/255$ and number of iterations as 7. We evaluate the D-SphereNets with different SphereCorrs and different learning obejctives.

**Black-box Attack.** For substitute models in blackbox attacks in both MNIST and CIFAR-10, we independently train both vanilla and D-SphereNet models with natural and FGSM training. The training data and parameters are kept the same to show the worst possible case.

## 7.2 WHITE-BOX ADVERSARIAL ATTACKS

**MNIST.** For each model with certain training method, we attack it with both FGSM and PGD. For FGSM attack, we use $\epsilon = 0.3$ as in the FGSM training. For PGD attack, we set step size as $0.01$ and the number of iterations as $100$. The results are shown in table 1.

With white-box attacks, vanilla CNN is prone to attacks on which the network is not trained on. On the contrary, variants of D-Spherenet are able to resist some portion of adversrial examples even though they are not explicitly trained on.

Among those trained with FGSM adversarial training, D-SphereNets with weight multiplier and loss annealing have the best performance against PGD attacks. Since loss annealing does not show significant advantage over weight multiplier, we will only experiment with weight multiplier in the following experiments.

**CIFAR-10.** For each model trained with each training method, we attack it with both FGSM and PGD. For FGSM attack, we use $\epsilon = 0.3$ (where the values of images range from 0 to 255) as in the FGSM training. For PGD attack, we use step size=0.01 and number of iterations=100. The result is shown in Table 2.

The results are mostly consistent with the observations in MNIST experiments, where D-Spherenets are able to classifier more adversarial examples than vanilla CNN.

Table 2: White-box attack on CIFAR-10 dataset.

| Attack | Baseline CNN | | D-SphereNet | | D-SphereNet + Weight Multiplier | |
|---|---|---|---|---|---|---|
| | Natural Training | FGSM Training | Natural Training | FGSM Training | Natural Training | FGSM Training |
| Natural Data | 0.8535 | 0.837 | 0.8858 | 0.8741 | 0.8797 | 0.884 |
| FGSM | 0.1882 | 0.7896 | 0.4364 | 0.8598 | 0.3834 | 0.869 |
| PGD | 0.0867 | 0.0796 | 0.0989 | 0.3507 | 0.2132 | 0.4443 |

| Attack | D-SphereNet + WM + Quad SphereCorr | | D-SphereNet + WM + Loss Annealing | | D-SphereNet + WM + Input-truncated SphereCorr | |
|---|---|---|---|---|---|---|
| | Natural Training | FGSM Training | Natural Training | FGSM Training | Natural Training | FGSM Training |
| Natural Data | 0.8851 | 0.8826 | 0.8964 | 0.8993 | 0.9214 | 0.9208 |
| FGSM | 0.4672 | 0.8673 | 0.4356 | 0.8939 | 0.3786 | 0.8951 |
| PGD | 0.2178 | 0.4669 | 0.1264 | 0.3811 | 0.0912 | 0.1728 |

Table 3: Blackbox attacks on CIFAR-10 dataset.

| Source Model | Attack | Target Model | | | | | |
|---|---|---|---|---|---|---|---|
| | | Vanilla | | D-SphereNet | | D-SphereNet + WM | |
| | | Natural Training | FGSM Training | Natural Training | FGSM Training | Natural Training | FGSM Training |
| Baseline CNN | FGSM | 0.5090 | 0.7757 | 0.5671 | 0.7629 | 0.5405 | 0.7468 |
| | PGD | 0.3622 | 0.7855 | 0.431 | 0.7779 | 0.3897 | 0.7599 |
| D-SphereNet | FGSM | 0.6877 | 0.7677 | 0.6725 | 0.8128 | 0.6505 | 0.8102 |
| | PGD | 0.7168 | 0.775 | 0.6946 | 0.836 | 0.6612 | 0.8415 |
| D-SphereNet + WM | FGSM | 0.674 | 0.765 | 0.6717 | 0.8102 | 0.6475 | 0.806 |
| | PGD | 0.7438 | 0.7798 | 0.7464 | 0.8457 | 0.7168 | 0.8458 |

D-SphereNet with both weight multiplier and trauncated projections is more capable of fitting both normal examples and FGSM generated examples than any other networks. We also notice that this variant network converges within only 25k iterations. However, this network, if trained with FGSM training, is not as robust as D-Spherenet with only weight multiplier.

### 7.3 Black-box Adversarial Attacks

We evaluate the robustness of D-SphereNets against black-box attacks on CIFAR-10. We use independently trained substitute networks to attack the D-Spherenets. In some experiments, the source model (substitute) and the target model have the same architecture. All the networks are trained on the same set of training data in order to examine the worst case performance of D-SphereNets. The results are shown in table 3.

Presumably, one can easily attack the target network with the knowledge of the architecture. This is not the case for D-SphereNets, however, as D-SphereNets are more robust against attacks on the D-SphereNets. We believe the source of robustness comes from the complex landscape of the loss function.

We also notice that, D-SphereNets have similar values of accuracy, compared to vanilla CNN, on adversarial examples generated by vanilla substitute CNN. This fact indicates that D-SphereNets cannot be specifically targeted while maintaining reasonable accuracy on adversarial examples generated with simple methods, with WM denoting weight multiplier.

## References

Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *ICML*, 2016.

Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *International Conference on Machine*

*Learning*, pp. 854–863, 2017.

Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Fundamental limits on adversarial robustness. In *Proc. ICML, Workshop on Deep Learning*, 2015.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Weiyang Liu, Yan-Ming Zhang, Xingguo Li, Zhiding Yu, Bo Dai, Tuo Zhao, and Le Song. Deep hyperspherical learning. In *NIPS*, 2017.

Jiajun Lu, Theerasit Issaranon, and David Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. *arXiv preprint arXiv:1704.00103*, 2017.

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. *arXiv preprint arXiv:1507.00677*, 2015.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *arXiv preprint arXiv:1610.08401*, 2016a.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2574–2582, 2016b.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 427–436, 2015.

Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v1. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016a.

Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pp. 372–387. IEEE, 2016b.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 582–597. IEEE, 2016c.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *CVPR*, 2014.

Stefan Wager, Sida Wang, and Percy S Liang. Dropout training as adaptive regularization. In *Advances in neural information processing systems*, pp. 351–359, 2013.

Stephan Zheng, Yang Song, Thomas Leung, and Ian Goodfellow. Improving the robustness of deep neural networks via stability training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4480–4488, 2016.

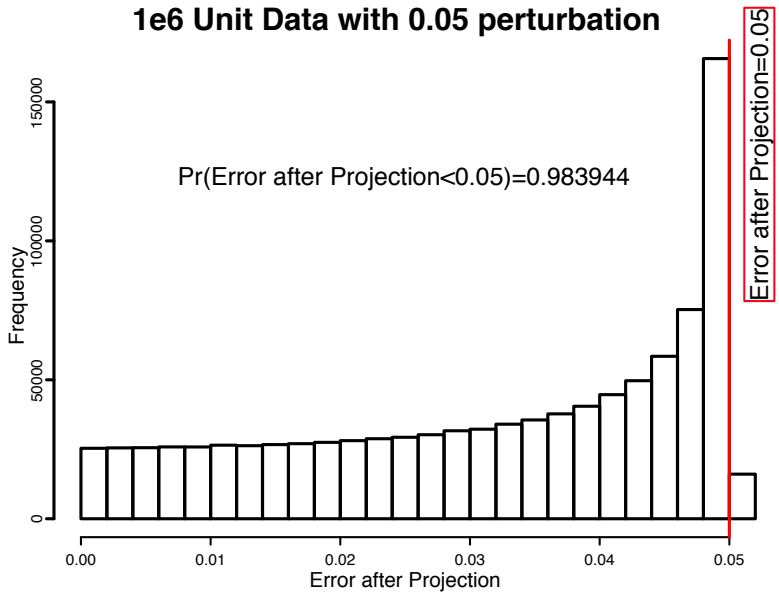



Figure 4: Histogram of Euclidean Error after Projecting the Adversarial Input

## A Illustration of Why Projection Works

Here, we draw a simple two-dimension example in Figure 4 to show that with high probability, projection does reduce the Euclidean Error caused by adversarial attack. The generating process is as follows: Fist, we uniformly sample one million points over $\mathcal{S}^1$. Without loss of generality, we choose $(0, 0.05)$ as the adversarial perturbation. After adding this to the whole sample, we project these adversarial examples to $\mathcal{S}^1$. Then we calculate the Euclidean distance between the original point and its projection point as its Euclidean Error. Next, we draw the histogram of the Euclidean Error. As we can see, projection seldom increases the Euclidean Error. For one million points in two dimensions, the probability of projection preserving the Error equals 0.98. This implies the function of projection.

## B Proof of the Theorem Justification

### B.1 Proof of Theorem 2

*Proof.* By assumption 1, we know that $\boldsymbol{x}$ is uniformly sampled from $\mathcal{S}^{p-1}$, the sphere of the unit ball in $\mathbb{R}^p$. Let $x_i$ and $\boldsymbol{w}_i^*$ denote the $i$-th entry of $\boldsymbol{x}$ and $\boldsymbol{w}^*$, respectively. Then we have

$$
\begin{aligned}
\mathbb{E}\left[(y - \widehat{\boldsymbol{x}}^\top \boldsymbol{w})^2\right] &= \mathbb{E}\left[(\boldsymbol{x}^\top \boldsymbol{w}^* - (\boldsymbol{x} - \delta \cdot \operatorname{sign}(\epsilon \boldsymbol{w}^*))^\top \boldsymbol{w} + \epsilon)^2\right] \\
&= \mathbb{E}\left[(\boldsymbol{x}^\top \boldsymbol{w}^* - \boldsymbol{x}^\top \boldsymbol{w})^2\right] + 2\mathbb{E}\left[(\boldsymbol{x}^\top (\boldsymbol{w}^* - \boldsymbol{w}))(\delta \operatorname{sign}(\epsilon \boldsymbol{w}^*)^\top \boldsymbol{w} + \epsilon)\right] + \mathbb{E}\left[(\delta \operatorname{sign}(\epsilon \boldsymbol{w}^*)^\top \boldsymbol{w} + \epsilon)^2\right] \\
&= \mathbb{E}[\sum_{i=1}^p x_i(\boldsymbol{w}_i^* - \boldsymbol{w}_i) \sum_{j=1}^p x_j(\boldsymbol{w}_j^* - \boldsymbol{w}_j)] + \delta^2 \left(\operatorname{sign}(\boldsymbol{w}^*)^\top \boldsymbol{w}\right)^2 + \sigma^2 + 2\mathbb{E}[\epsilon\delta \operatorname{sign}(\epsilon \boldsymbol{w}^*)^\top \boldsymbol{w}] \\
&= \mathbb{E}[\sum_{i=1}^p x_i^2(\boldsymbol{w}_i^* - \boldsymbol{w}_i)^2] + 2\sqrt{\frac{2}{\pi}}\delta\sigma \operatorname{sign}(\boldsymbol{w}^*)^\top \boldsymbol{w} + \delta^2 \left(\operatorname{sign}(\boldsymbol{w}^*)^\top \boldsymbol{w}\right)^2 + \sigma^2, \text{(by symmetric of } x_i) \\
&= \frac{1}{p}\|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 + \left(\delta \operatorname{sign}(\boldsymbol{w}^*)^\top \boldsymbol{w} + \sqrt{\frac{2}{\pi}}\sigma\right)^2 + \frac{\pi - 2}{\pi}\sigma^2.
\end{aligned}
\tag{16}
$$

The last equality holds because we can obtain $\mathbb{E}x_i^2 = \frac{1}{p}$ by $\mathbb{E}\sum_{i=1}^p x_i^2 = 1$ and the isotropic of $x_i$'s. Also, since both $\delta$ and $\sigma$ are sufficiently small, the first term of Right Hand Side (RHS) in (16) dominates the whole loss. This leads the optimal solution, i.e. $\widehat{\boldsymbol{w}}_1$, is close to $\boldsymbol{w}^*$, which implies that

$$\operatorname{sign}(\boldsymbol{w}^*)^\top \widehat{\boldsymbol{w}}_1 \approx \|\widehat{\boldsymbol{w}}_1\|_1 .$$

Therefore, we have the following approximate solution for $\widehat{\boldsymbol{w}}_1$,

$$\widehat{\boldsymbol{w}}_1 \approx \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{1}{p} \|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2 + \left(\delta \|\boldsymbol{w}\|_1 + \sqrt{\frac{2}{\pi}}\sigma\right)^2.$$

□

### B.2 PROOF OF THEOREM 3

*Proof.* For notation simplicity, let $a = \operatorname{sign}(\epsilon),\ \boldsymbol{b} = \operatorname{sign}(\boldsymbol{w}^*)$. Note $a^2 = 1,\ \boldsymbol{b}^\top \boldsymbol{b} = p$. We obtain

$$\begin{aligned}\frac{1}{\|\widetilde{\boldsymbol{x}}\|_2}\widetilde{\boldsymbol{x}} &= \frac{1}{\sqrt{\left(1 - 2\delta \cdot a\boldsymbol{b}^\top \boldsymbol{x} + \delta^2 \boldsymbol{b}^\top \boldsymbol{b}\right)}}\widetilde{X} \\ &= \left[1 - \frac{1}{2}(-2\delta a \boldsymbol{b}^\top \boldsymbol{x} + p\delta^2)\right](\boldsymbol{x} - \delta a \boldsymbol{b}) + O(\delta^2) \\ &= \boldsymbol{x} + \delta\left(a\boldsymbol{b}^\top \boldsymbol{x} \cdot \boldsymbol{x} - a\boldsymbol{b}\right) + O(\delta^2).\end{aligned} \tag{17}$$

The second equality holds because of Taylor Expansion. We plug (17) into the objective function as follows: (if not clear specified, we denote $\mathbb{E}_{X,\epsilon}$ by $\mathbb{E}$ for notational simplicity)

$$\begin{aligned}&\mathbb{E}\left[(y - \frac{1}{\|\widetilde{\boldsymbol{x}}\|_2}\widetilde{\boldsymbol{x}}^\top \boldsymbol{w})^2\right] \\ \approx&\mathbb{E}\left[(\boldsymbol{x}^\top \boldsymbol{w}^* - \left(\boldsymbol{x} + \delta\left(a\boldsymbol{b}^\top \boldsymbol{x} \cdot \boldsymbol{x} - a\boldsymbol{b}\right)\right)^\top \boldsymbol{w} + \epsilon)^2\right] \\ =&\mathbb{E}\Big[\big(\underbrace{\boldsymbol{x}^\top(\boldsymbol{w}^* - \boldsymbol{w})}_{A} + \underbrace{\epsilon - \delta a \boldsymbol{b}^\top \boldsymbol{x} \cdot \boldsymbol{x}^\top \boldsymbol{w} + \delta a \boldsymbol{b}^\top \boldsymbol{w}}_{B}\big)^2\Big] \\ =&\mathbb{E}\left[A^2 + 2A \cdot B + B^2\right].\end{aligned} \tag{18}$$

Let $x_i,\ b_i,\ \boldsymbol{w}_i$ denote $i$-th component of $\boldsymbol{x},\ \boldsymbol{b},\ \boldsymbol{w}$ respectively. Now, let's calculate the terms of RHS in (18) one by one:

$$\mathbb{E}\left[A^2\right] = \mathbb{E}\left[\boldsymbol{x}^\top(\boldsymbol{w}^* - \boldsymbol{w})\right] = \frac{1}{p}\|\boldsymbol{w} - \boldsymbol{w}^*\|_2^2, \quad \text{(from (16))}; \tag{19}$$

$$\begin{aligned}\mathbb{E}\left[A \cdot B\right] &= \mathbb{E}_{\boldsymbol{x}}\left[\left(\boldsymbol{x}^\top(\boldsymbol{w}^* - \boldsymbol{w})\mathbb{E}_\epsilon(\epsilon - \delta a \boldsymbol{b}^\top \boldsymbol{x} \cdot \boldsymbol{x}^\top \boldsymbol{w} + \delta a \boldsymbol{b}^\top \boldsymbol{w})\right)\right] \\ &= 0, \quad \text{(notice } \mathbb{E}_\epsilon a = \mathbb{E}_\epsilon \epsilon = 0, \boldsymbol{x} \text{ and } \epsilon \text{ are independent)};\end{aligned} \tag{20}$$

$$\begin{aligned}\mathbb{E}\left[B^2\right] &= \mathbb{E}\left[\left(\epsilon - \delta a \boldsymbol{b}^\top \boldsymbol{x} \cdot \boldsymbol{x}^\top \boldsymbol{w} + \delta a \boldsymbol{b}^\top \boldsymbol{w}\right)^2\right] \\ &= \mathbb{E}\Big[\underbrace{(\epsilon + = \delta a \boldsymbol{b}^\top \boldsymbol{w})^2}_{C} - 2\underbrace{(\delta^2 \boldsymbol{w}^\top \boldsymbol{b} + \epsilon a)\boldsymbol{b}^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{w}}_{D} + \underbrace{\delta^2(\boldsymbol{b}^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{w})^2}_{E}\Big];\end{aligned} \tag{21}$$

Like before, we calculate the each part of RHS in (21) as follows:

$$\mathbb{E}\left[C\right] = \mathbb{E}_\epsilon\left[\epsilon^2 + 2\delta\epsilon a \boldsymbol{b}^\top \boldsymbol{w} + \delta^2(\boldsymbol{b}^\top \boldsymbol{w})^2\right] = \sigma^2 + 2\sqrt{\frac{2}{\pi}}\delta\sigma \boldsymbol{b}^\top \boldsymbol{w} + \delta^2(\boldsymbol{b}^\top \boldsymbol{w})^2; \tag{22}$$

$$\mathbb{E}\left[D\right] = \mathbb{E}_\epsilon\left[\delta^2 \boldsymbol{w}^\top \boldsymbol{b} + \epsilon a\delta\right]\mathbb{E}_{\boldsymbol{x}}\left[\boldsymbol{b}^\top \boldsymbol{x}\boldsymbol{x}^\top \boldsymbol{w}\right] = \left(\delta^2 \boldsymbol{w}^\top \boldsymbol{b} + \sqrt{\frac{2}{\pi}}\delta\sigma\right)\frac{1}{p}\boldsymbol{b}^\top \boldsymbol{w}; \tag{23}$$

$$\begin{aligned}\mathbb{E}\left[E\right] &= \delta^2 \mathbb{E}_{\boldsymbol{x}}\Big[\sum_{i,j,k,l=1}^{p} b_i x_i x_j \boldsymbol{w}_j b_k x_k x_l \boldsymbol{w}_l\Big] \\ &= \delta^2 \mathbb{E}_{\boldsymbol{x}}\Big[\sum_{i,j=1, i\neq j}^{p} x_i^2 x_j^2 \boldsymbol{w}_j^2 + 2\sum_{i,k=1, i\neq k}^{p} b_i x_i^2 \boldsymbol{w}_i b_k x_k^2 \boldsymbol{w}_k + \sum_{i=1}^{p} x_i^4 \boldsymbol{w}_i^2\Big].\end{aligned} \tag{24}$$

To simplify (24), let's consider a random vector $\boldsymbol{Z}$. $Z_i$, the $i$-th entry of $\boldsymbol{Z}$, is defined as follows:

$$Z_i := \frac{z_i}{\sqrt{\sum_{j=1}^{p} z_j^2}}, \quad \text{where } z_i \sim N(0, 1).$$

Note that actually $\boldsymbol{Z}$ follows a uniform distribution over $\mathcal{S}^{p-1}$ and $Z_i^2 \sim B(\frac{1}{2}, \frac{p-1}{2})$. By this, we calculate $\mathbb{E}\big[x_i^4\big]$ in the following way:

$$\mathbb{E}\big[x_i^4\big] = \mathbb{E}(Z_i^2)^2 = \big(\mathrm{Var}(Z_i^2) + (\mathbb{E}Z_i^2)^2\big) = \left(\frac{\frac{1}{2}\frac{p-1}{2}}{(\frac{p}{2})^2(\frac{p+2}{2})} + (\frac{\frac{1}{2}}{\frac{1}{2}+\frac{p-1}{2}})^2\right) = \frac{3}{(p+2)p}.$$

By $\mathbb{E}(\sum_{i=1}^p x_i^2)^2 = 1$ and symmetric of $x_i$, we get $\mathbb{E}x_i^2x_j^2 = \frac{1}{(p+2)p}$, $\forall i \neq j$. Plug these into (24).

$$\begin{aligned}\mathbb{E}\big[E\big] &= \delta^2\big[\frac{1}{(p+2)p}\sum_{i,j=1,i\neq j}\boldsymbol{w}_j^2 + 2\frac{1}{(p+2)p}\sum_{i,j=1,i\neq j} b_ib_j\boldsymbol{w}_j\boldsymbol{w}_i + \frac{3}{(p+2)p}\sum_{i=1}^p \boldsymbol{w}_i^2\big] \\ &= \delta^2\big[\frac{1}{p+2}\left\|\boldsymbol{w}\right\|_2^2 + \frac{2}{p(p+2)}\left\|\boldsymbol{w}\right\|_1^2\big]. \end{aligned} \tag{25}$$

Combine (19), (21), (22), (23), (25) together, we obtain

$$\begin{aligned}(18) = &\frac{1}{p}\left\|\boldsymbol{w}-\boldsymbol{w}^*\right\|_2^2 + \sigma^2 + 2\sqrt{\frac{2}{\pi}}\delta\sigma\boldsymbol{b}^\top\boldsymbol{w} + \delta^2(\boldsymbol{b}^\top\boldsymbol{w})^2 \\ &- 2\Big(\big(\delta^2\boldsymbol{w}^\top\boldsymbol{b} + \sqrt{\frac{2}{\pi}}\delta\sigma\big)\frac{1}{p}\boldsymbol{b}^\top\boldsymbol{w}\Big) + \delta^2\big[\frac{1}{p+2}\left\|\boldsymbol{w}\right\|_2^2 + \frac{2}{p(p+2)}\left\|\boldsymbol{w}\right\|_1^2\big].\end{aligned}$$

Note that both $\delta,\ \sigma$ are small enough, the dominant term is $\frac{1}{p}\left\|\boldsymbol{w}^*-\boldsymbol{w}\right\|_2^2$. This implies

$$\mathrm{sign}(\boldsymbol{w}^*)^\top\widehat{\boldsymbol{w}}_2 \approx \left\|\widehat{\boldsymbol{w}}_2\right\|_1.$$

By this approach, we further approximate $\widehat{\boldsymbol{w}}_2$ as follows:

$$\widehat{\boldsymbol{w}}_2 \approx \operatorname*{argmin}_{\boldsymbol{w}} \frac{1}{p}\left\|\boldsymbol{w}-\boldsymbol{w}^*\right\|_2^2 + \frac{p-2}{p}\big(\delta\left\|\boldsymbol{w}\right\|_1 + \sqrt{\frac{2}{\pi}}\sigma\big)^2 + \delta^2\big(\frac{1}{p+2}\left\|\boldsymbol{w}\right\|_2^2 + \frac{2}{p(p+2)}\left\|\boldsymbol{w}\right\|_1^2\big).$$

□

### B.3 Proof of Theorem 5

*Proof.* First, let's calculate the Left Hand Side (LHS) in (15).

$$\begin{aligned}\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\widetilde{\boldsymbol{x}}^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2 &= \mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\widehat{\boldsymbol{x}}^\top\boldsymbol{w}^* - \frac{\delta}{\sqrt{p}}\,\mathrm{sign}(\epsilon)\,\mathrm{sign}(\boldsymbol{w}^*)^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2 \\ &= \mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|-\frac{\delta}{\sqrt{p}}\,\mathrm{sign}(\boldsymbol{w}^*)^\top\boldsymbol{w}^*\right\|_2^2 = \frac{\delta^2}{p}\left\|\boldsymbol{w}^*\right\|_1^2,\end{aligned}$$

where $\epsilon$ does not affect the value at all. For the projected input, by (17), (25) and (23), we have

$$\begin{aligned}\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\frac{1}{\left\|\widetilde{\boldsymbol{x}}\right\|_2}\widetilde{\boldsymbol{x}}^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2 &= \frac{\delta^2}{p}\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\mathrm{sign}(\boldsymbol{w}^*)^\top\widehat{\boldsymbol{x}}\cdot\widehat{\boldsymbol{x}}^\top\boldsymbol{w}^* - \mathrm{sign}(\boldsymbol{w}^*)^\top\boldsymbol{w}^*\right\|_2^2 \\ &= \frac{\delta^2}{p}\big[\frac{1}{p+2}\left\|\boldsymbol{w}^*\right\|_2^2 + \frac{2}{p(p+2)}\left\|\boldsymbol{w}^*\right\|_1^2 - \frac{2}{p}\left\|\boldsymbol{w}^*\right\|_1^2 + \left\|\boldsymbol{w}^*\right\|_1^2\big] \\ &\leq \frac{\delta^2}{p}\left\|\boldsymbol{w}^*\right\|_1^2 - \frac{\delta^2}{p^2}\left\|\boldsymbol{w}^*\right\|_1^2 = \frac{p-1}{p}\mathbb{E}_{\widehat{\boldsymbol{x}}}\left\|\widetilde{\boldsymbol{x}}^\top\boldsymbol{w}^* - \widehat{\boldsymbol{x}}^\top\boldsymbol{w}^*\right\|_2^2.\end{aligned}$$

where $\epsilon$ does not affect the value, too. And we prove the desired result. □

## C Experimental Settings

| Layer | CNN for MNIST | CNN for CIFAR10 |
|---|---|---|
| Conv1.x | [3×3, 32]×2 | [3×3, 64]×3 |
| Pool1 | 2×2 Max, Stride 2 | |
| Conv2.x | [3×3, 64]×2 | [3×3, 128]×3 |
| Pool2 | 2×2 Max, Stride 2 | |
| Conv3.x | [3×3, 128]×2 | [3×3, 256]×3 |
| Pool3 | 2×2 Max, Stride 2 | |
| Fully Connected | 256 | 512 |

Table 4: Our CNN architectures for different benchmark datasets. Conv1.x, Conv2.x and Conv3.x denote convolution units that may contain multiple convolution layers. E.g., [3×3, 64]×3 denotes 3 cascaded convolution layers with 64 filters of size 3×3.