arXiv:2008.12338v1 [cs.LG] 27 Aug 2020

Adversarially Robust Learning via Entropic Regularization

Gauri Jagatap, Animesh Basak Chowdhury, Siddharth Garg and Chinmay Hegde
New York University
{gauri.jagatap,abc586,sg175,chinmay.h}@nyu.edu

Abstract

In this paper we propose a new family of algorithms for training adversarially robust deep neural networks. We formulate a new loss function that uses an entropic regularization. Our loss function considers the contribution of adversarial samples which are drawn from a specially designed distribution that assigns high probability to points with high loss from the immediate neighborhood of training samples. Our data entropy guided SGD approach is designed to search for adversarially robust valleys of the loss landscape. We observe that our approach generalizes better in terms of classification accuracy robustness as compared to state of art approaches based on projected gradient descent.

1 Introduction

Deep neural networks have led to significant breakthroughs in the fields of computer vision [1], natural language processing [2], speech processing [3], recommendation systems [4] and forensic imaging [5]. However, deep networks have also been shown to be very susceptible to carefully designed "attacks" [6, 7, 8]. In particular, the outputs of networks trained via traditional approaches are rather brittle to maliciously crafted perturbations in both input data as well as network weights [9].

Formally put, suppose the forward map between the inputs x and outputs y is modelled via a neural network as $y = f(w; x)$ where w represents the set of trainable weight parameters. For a classification task, given a labeled dataset $\{x_i, y_i\}$, $i = 1, \dots, n$, the standard procedure for training neural networks is to seek the weight parameters w that minimize the empirical risk:

$$\hat{w} = \arg\min_{w} \frac{1}{n} \sum_{i=1}^{n} L(f(w; x_i), y_i) := \mathcal{L}(X; Y, w).$$

However, the prediction $\hat{y}(x) = f(\hat{w}; x)$ can be very sensitive to changes in both w and x. For example, if a bounded perturbation to a test image input (or to the neural network weights) is permitted, i.e. $\hat{y}_i = f(w; x_i + \delta_i)$ where δ_i represents the perturbation, then the predicted label $\hat{y}_i$ can be made *arbitrarily* different from the true label y_i.

Several techniques for finding such adversarial perturbations have been put forth. Typically, this can be achieved by maximizing the loss function within a neighborhood around the test point x [10, 11]:

$$\bar{x}_{\text{worst}} = \arg\max_{\delta \in \Delta_p} L(f(\hat{w}; x + \delta), y) \quad (1)$$

The perturbation set Δ_p is typically an ℓ_p ball for some $p \in \{0, 1, 2, \infty\}$. We discuss several attack models in the next section.

The existence of adversarial attacks motivates the need for a "defense" mechanism that makes the network under consideration more robust. Despite a wealth of proposed defense techniques, the jury is still out on how optimal defenses should be constructed [12]. Broadly, there seem to be two

Preprint. Under review.

families of effective defenses. The first involves *adversarial training* [11]. Here, an adversarial input is discovered by solving (16) and added to the training dataset; this has the effect of optimizing a robust version of the empirical risk:

$$\hat{w} = \min_{w} \max_{\delta \in \Delta_p} \frac{1}{n} \sum_{i=1}^{n} L(f(w; x_i + \delta), y_i).$$

The second involves *randomized smoothing*. Here, both training the network as well as the inference made by the network are smoothed out over several stochastic perturbations of the attack example [13, 14, 15]. This has the effect of optimizing a smoothed-adversarial version of the empirical risk.[1]

In this paper, we propose a new approach for learning adversarially robust neural networks. Our key conceptual ingredient underlying our approach is *entropic regularization*. Borrowing intuition from [16], instead of the empirical risk (or its adversarial counterpart), our algorithm instead optimizes over an local entropy-regularized version of the empirical risk:

$$\hat{w} = \arg\min_{w} \left(\int_{X'} \mathcal{L}(X'; Y, w) \left[Z^{-1} \exp\left(\mathcal{L}(X'; Y, w) - \frac{\gamma}{2} \|X - X'\|_p^p \right) \right] \right) dX'. \qquad (2)$$

Intuitively, this new loss function can be viewed as the convolution of the empirical risk with a specific Gibbs distribution to sample points from the neighborhoods X' of the training data points X. Therefore, compared to adversarial training, we have replaced the inner maximization with an expected value with respect to the Gibbs measure which is matched to the *geometry* of the perturbation set.

Since the above loss function is difficult to optimize (or even evaluate exactly), we instead approximate it via Monte Carlo techniques. In particular, we use Stochastic Gradient Langevin Dynamics [17], which is well-known to be scalable to large datasets. In this manner, our approach blends in elements from both adversarial training as well as randomized smoothing.

To summarize, our specific contributions are as follows:

1. We propose a new entropy-regularized loss function for training deep neural networks (Eq. 2) that is a robust version of the empirical risk.
2. We propose a new Monte Carlo algorithm to optimize this new loss function that is based on Stochastic Gradient Langevin Dynamics. We call this approach *Data-Entropy SGD* (DESGD).
3. We show that DESGD-trained networks provide improved (robust) test accuracy when compared to existing defense approaches.

In particular, we are able to train an ℓ_∞-robust CIFAR-10 model to 41.8% accuracy at attack level $\epsilon = 8/255$, which is higher than the latest reported levels for defenses based on both adversarial training as well as randomized smoothing [11, 18, 15].

2 Relation to Prior Work

Due to tight space constraints, our review of related work will unfortunately be incomplete. See [19] for a more thorough review of the literature.

Evidence for the existence of adversarial inputs for deep neural networks is by now well established [20, 21, 22, 23, 24, 25]. Let us focus on image classification. The majority of attacks have focused on the setting where the adversary confounds the classifier by adding an *imperceptible* perturbation to a given input image. The range of the perturbation is pre-specified in terms of bounded pixel-space ℓ_p-norm balls. Specifically, an ℓ_p- attack model allows the adversary to search over the set of input perturbations $\Delta_{p,\epsilon} = \{\delta : \|\delta\|_p \leq \epsilon\}$ for $p = \{0, 1, 2, \infty\}$.

Initial attack methods, including the Fast Gradient Sign Method (FGSM) and its variants [26, 27, 28]. proposed techniques for generating adversarial examples by ascending along the sign of the loss gradient:

$$x_{adv} = x + \epsilon \,\mathrm{sgn}(\nabla_x L(f(\hat{w}; x), y)),$$

[1]The latter family of methods has the additional benefit of being *certifiably robust*: all points within a ball of a given radius around the test point are provably classified with the correct label.

where $(x_{adv} - x) \in \Delta_{\infty,\epsilon}$. Madry *et. al.* [11] proposed a stronger adversarial attack via projected gradient descent (PGD) by iterating FGSM several times, such that

$$x^{t+1} = \Pi_{x+\Delta_{p,\epsilon}}(x^t + \alpha \operatorname{sgn}(\nabla_x L(f(\hat{w}; x), y)),$$

where $p = \{2, \infty\}$. These attacks are (arguably) the most successful available attack techniques reported to date, and serve as the basis of our comparisons.

For given attack models, several defense strategies have been developed. In [11], adversarial training is performed via the min-max formulation Eq. 16. The inner maximization is solved using PGD, while the outer objective is minimized using stochastic gradient descent (SGD) with respect to w. This can be slow to implement, and speed-ups have been proposed in [29] and [18]. In [30, 14, 13, 15], the authors developed certified defense strategies via randomized smoothing. This approach consists of two stages: the first stage consists of training with noisy samples, and the second stage produces an ensemble-based inference. [19] contains a detailed overview of attack and defense models.

In a different line of work, there have been efforts towards building neural network networks with improved generalization properties. In particular, heuristic experiments by [31, 32, 33] suggest that the loss surface at the final learned weights for well-generalizing models is relatively “flat". Building on this intuition, Chaudhari *et. al.* [16] showed that by explicitly introducing a smoothing term (via entropic regularization) to the training objective, the learning procedure weights towards regions with flatter minima by design. Their approach, Entropy-SGD (or ESGD) is shown to possess better generalization properties in deep networks. We leverage this intuition, but develop a new algorithm for training deep networks with better *adversarial robustness* properties.

3 Problem Formulation

The task of classification, given a training labelled dataset $\{x_i \in \mathcal{X}, y_i\}$, $i \in \{1, \ldots, n\}$, consists of solving the standard objective by optimizing weight parameters w, such that $\min_w \frac{1}{n} \sum_{i=1}^{n} L(f(w; x_i), y_i)$, where y_i is a one-hot class encoding vector of length m and m is the total number of classes. The training data matrix itself is stored as $X \in \mathbb{R}^{n \times d}$ and labels in $Y \in \mathbb{R}^n$ where we have access to n training samples which are d-dimensional each. Given this formulation, the primary task is to minimize the Cross Entropy Loss (CEL) function $\mathcal{L}$, which can be defined as follows:

$$\mathcal{L}(w; X, Y) = -\frac{1}{n} \sum_{i=1}^{n} (1 - y_i) \log(1 - \hat{y_i}) + y_i \log(\hat{y_i}). \tag{3}$$

Several existing optimization methods from the literature can be used to perform this minimization, including Stochastic Gradient Descent (SGD), ADAM, and Adagrad. In this paper, we design an augmented version of the loss function in Eq. 3, and also introduce a procedure that we call Data-Entropy SGD to minimize the same. We first recap the Entropy SGD [16]; see also Appendix A of the supplement. Entropy-SGD considers an augmented loss function of the form:

$$\mathcal{L}_{ent}(w; X, Y) = -\log \left(\int_{w'} \exp \left(-\mathcal{L}(w'; X, Y) - \frac{\gamma}{2} \|w - w'\|_2^2 \right) dw' \right) \tag{4}$$

By design, minimization of this augmented loss function promotes minima with wide valleys. Such a minima would be robust to perturbations in w, however may not be necessarily be advantageous against adversarial data samples x_{adv} (see Section 4). In our experiments section below we show that Entropy-SGD offers comparable (or even worse) robustness to adversarial attacks compared to standard SGD.

For the task of adversarial robustness, we instead develop a data-dependent version of Entropy-SGD. To model for perturbations in data instead of weights, we design an augmented loss that regularizes the *data* space. Note that we are only looking at specific perturbations that *increase* the overall loss value of prediction. In order to formally motivate our approach, we first make some assumptions.

Assumption 1. *The distribution of possible adversarial data inputs of the neural network obeys a positive exponential distribution of the form below, where the domain of $\mathcal{L}(X; Y, w)$ is bounded:*

$$p(X; Y, w, \beta) = \begin{cases} Z_{w,\beta}^{-1} \exp\left(\beta \mathcal{L}(X; Y, w)\right) & \textit{if} \quad 0 \le \mathcal{L}(X; Y, w) \le R, \\ 0 & \textit{if} \quad \mathcal{L}(X; Y, w) > R, \end{cases} \tag{5}$$

and $Z_{w,\beta}$ is the partition function that normalizes the probability distribution.

Intuitively, the neural network is more likely to "see" perturbed examples from the adversary corresponding to higher loss values as compared to lower loss values. (The parameter R is to chosen to ensure that the integral of the probability curve is bounded.) When the temperature parameter $\beta \to \infty$, the above Gibbs distribution concentrates at the maximizer(s) of $\mathcal{L}(\bar{X}; Y, w)$, where $\bar{X}$ is the "worst possible" set of adversarial inputs to the domain of the loss function for fixed weights w.

Assumption 2. *Set $\beta = 1$. A modified distribution, (without loss of generality, setting $\beta = 1$) with an additional smoothing parameter, assumes the form:*

$$p(X'; X, Y, w, \gamma) = \begin{cases} Z^{-1}_{X,w,\gamma} \exp\left(\mathcal{L}(X'; Y, w) - \frac{\gamma}{2}\|X' - X\|_F^2\right) & \text{if} \quad 0 \le \mathcal{L}(X'; Y, w) \le R \\ 0 & \text{if} \quad \mathcal{L}(X'; Y, w) > R \end{cases} \tag{6}$$

where $Z_{X,w,\gamma}$ is the partition function that normalizes the probability distribution.

Here γ controls the penalty of the distance of the adversary from true data X; if $\gamma \to \infty$, the sampling is sharp, and this corresponds to no smoothing effect, which is the same as minimizing the standard loss, meanwhile $\gamma \to 0$ corresponds to a uniform contribution from all possible data points in the loss manifold.

Now, we develop an augmented loss function which incorporates the probabilistic formulation in Assumption 3. The standard objective can be re-written as the functional convolution:

$$\min_w \mathcal{L}(w; X, Y) := \min_w \left(\int_{X'} \mathcal{L}(X'; Y, w) \delta(X - X') dX' \right)$$

which can be seen as a sharp sampling of the loss function at training points X. Now, if we define the *Data-Entropy Loss* as:

$$\mathcal{L}_{de}(w; X, Y) = \int_{X'} \mathcal{L}(X'; Y, w) p(X'; X, Y, w, \gamma) dX' \tag{7}$$

our new objective is to minimize this augmented objective function $\mathcal{L}_{de}(w; x)$, which resembles expected value of the standard loss function sampled according to a distribution that (i) penalizes points further away from the true training data (ii) boosts data points which correspond to high loss values. Specifically, the adversarial samples generated by the distribution in Assumption 3 will correspond to those with high loss values in the immediate neighborhood of the true data samples.

If gradient descent is used to minimize the loss in Eq. 7, the gradient update corresponding to the augmented loss function can be computing as follows

$$\begin{aligned} \nabla_w \mathcal{L}_{de}(w; X, Y) &= \nabla_w \int_{X'} \mathcal{L}(X'; Y, w) p(X'; X, Y, w, \gamma) dX' \\ &= \nabla_w \mathbb{E}_{X' \sim p(X')}[\mathcal{L}(X'; Y, w)] \end{aligned} \tag{8}$$

Correspondingly, the weights of the network, when trained using gradient descent, using Eq. 8, can be updated as

$$w^+ = w - \eta \nabla_w \mathcal{L}_{de}(w; X, Y) \tag{9}$$

where η is the step size. The expectation in Eq. 8 is carried out over the probability distribution of data samples X' as defined in Assumption 2. This can be seen as the adversarial version of the formulation developed in Entropy SGD [16], where the authors use a Gibbs distribution to model an augmented loss function that explores the loss surface at points that are perturbed from the current weights w, denoted by w' (see Appendix A of the supplement). In contrast, in our approach, we consider loss contributions from perturbations X' of data points X. This analogue is driven by the fact that the core objective in [16] is to design a network which is robust to perturbations in *weights*, where as the core objective of this paper is to design a network that is robust to perturbations in the *inputs*.

The expectation in Eq. 8 is computationally intractable to optimize (or evaluate). However, using the Euler discretization of the Langevin Stochastic Differential Equation [17], it can be approximated well. Samples can be generated from $p(X')$ as:

$$X'^{t+1} = X'^{t} + \eta' \nabla_{X'} \log p(X'^{t}) + \sqrt{2\eta'} \varepsilon \mathcal{N}(0, \mathbb{I}) \tag{10}$$

Algorithm 1 Data Entropy SGD

```
1: Input: $X = [X_{B_1}, X_{B_2} \dots X_{B_J}], f, \eta, \eta', w = w^0, \gamma, \varepsilon$
2: for $t = 0, \cdots T-1$ do
     (outer loop of SGD)
3:   for $j = 1, \cdots J$ do
       (scan through all batches of data)
4:     $x_i^0 \leftarrow x_i + \delta_i$ {$\forall x_i \in X_{B_j}$, $L$ is number of samples generated using Langevin dynamics}
5:     $\mu^j \leftarrow 0$
6:     for $k = 0, \cdots, L-1$ do
7:       $dx'^k \leftarrow \frac{1}{n_j}\sum_{i=1}^{n_j} \nabla_{x=x'^k} L(f(w^t; x)) + \gamma(x^k - x'^k)$
8:       $x'^{k+1} \leftarrow x'^k + \eta' dx'^k + \sqrt{2\eta'}\varepsilon\mathcal{N}(0,1)$ {Langevin update}
9:       $\mu^k \leftarrow \frac{1}{B}\sum_{x_i \in X_{B_j}} L(w^t; x'^{k+1})$ {augmented batch loss for $X_{B_j}$}
10:      $\mu^j \leftarrow \frac{1}{L}(\mu^j + \mu^k)$
11:    end for
12:    $dL^t \leftarrow \nabla_w \mu^j$
13:    $w^{t+1} \leftarrow w^t - \eta dL^t$
14:  end for
15: end for
16: Output $\hat{w} \leftarrow w^T$
```

where η' is the step size for Langevin sampling, ε is a scaling factor that controls additive noise. In Langevin dynamics, when one considers the starting point of $X'^0 \sim \mathcal{N}(0, \mathbb{I})$ then the procedure above yields $X'^{t\to\infty}$ that follows the distribution $p(X')$. Intuitively, the stochastic process X'^t is more likely to visit points in the immediate neighborhoods of the *entire training* dataset X corresponding to high loss values.

Observe that X' and X have the same dimensions and the gradient term in the above equation needs to be computed over n, d-dimensional data points. In practice this can be computationally expensive. Therefore, we discuss a stochastic variant of this update rule, which considers mini-batches of training data instead, in a subsequent subsection. Plugging in the distribution in Eq.5, and using the Euler discretization for Langevin Stochastic Differential Equations, the update rule for sampling X' is

$$\begin{aligned} X'^{t+1} &= X'^t + \eta\nabla_{X'^t}\left(\mathcal{L}(X'^t; Y, w) - \frac{\gamma}{2}\|X - X'^t\|_F^2\right) + \sqrt{2\eta}\mathcal{N}(0, \mathbb{I}) \\ &= X'^t + \eta\nabla_{X'}\mathcal{L}(X'^t; Y, w) + \gamma(X - X'^t) + \sqrt{2\eta}\mathcal{N}(0, \mathbb{I}) \end{aligned} \tag{11}$$

where we have incorporated $Z_{X,w\gamma}$ in the step size η. Note that as the number of updates $t \to \infty$, the estimates from the procedure in Eq. 11 converge to samples from the true distribution $p(X')$. We then want to estimate estimate $\nabla_w \mathcal{L}_{de}(w; X, Y) = \nabla_w \mathbb{E}_{X'\sim p(X')}\left[\mathcal{L}(w; X', Y)\right]$ using this sampling procedure.

Since $X'^\infty \sim p(X')$, the expected value of a function of X' can be derived by simply using a sample mean of the sequence of iterates produced by the Langevin procedure [17] as is typical in the Markov Chain Monte Carlo (MCMC) literature. One can estimate $\mathbb{E}_{X'\sim p(X')}\left[\mathcal{L}(w; X', Y)\right]$ by averaging over many such iterates generated from this process.

Batch-wise updates for stochastic gradient estimates: As is typical with large datasets, instead of using the entire training data for computing gradients in Eq. 8 and Eq. 10, one can use batch-wise data where the training data is segmented into J batches $[X_{B_1}, X_{B_2} \dots X_{B_J}]$. This is essentially a combination of Stochastic Gradient Descent and Langevin Dynamics and is known as Stochastic Gradient Langevin Dynamics in recent literature [17]. We discuss this adaptation in further detail in Appendix B in the supplementary material.

This discussion effectively leads to the algorithm shown in Algorithm 1, which we refer to as Data Entropy SGD (or DE-SGD).

Comparison to PGD-SGD: In [11] the authors use adversarial training to produce robust networks from scratch against universal first order attacks generated using PGD. This setup is described in [11] (see also Algorithm 3 in Appendix C of the supplementary material for reference) and we will call this PGD-SGD.

It is easy to see that the updates of the robust training algorithm in [11] (Algorithm 3, Appendix C, supplement) are similar to that of Algorithm 1, consisting broadly of two types of gradient operations in an alternating fashion - (i) an (inner) gradient with respect to samples X (or batch-wise samples X_{B_j}) and (ii) an (outer) gradient with respect to weights w. While PGD-SGD minimizes the *worst-case* loss in an ϵ-neighborhood (specifically ℓ_2 or ℓ_∞ ball) of X, DESGD minimizes an *average loss* over our specifically designed probability distribution (Assumption 3) in the neighborhood of X. DESGD does this via sampling points in the immediate ℓ_2 neighborhood of X, which correspond to high loss values, with higher probability as compared to points further away.

The width of the Gaussian smoothing is adjusted with γ, which is analogous to controlling the projection radius ϵ in the inner-maximization of PGD-SGD (refer Algorithm 3 in Appendix C of the supplement). The Langevin sampling updates in Algorithm 1 are therefore analogous to the gradient updates in the inner maximization of Algorithm 3 in Appendix C, with an additional distance annealing penalty and noise. Meanwhile, there is an outer minimization of an augmented loss function with respect to network weights in both algorithms. In this way, DESGD can be re-interpreted as a stochastic formalization of ℓ_2-PGD-SGD, with noisy controlled updates. Intuitively this also suggests that the formulation described in Eq. 11 should be more robust against ℓ_2-ball attacks.

Extension to defense against ℓ_∞-attacks: It is evident that due to the isotropic structure of the Gibbs measure around each data point, the primary form of DESGD is best suited for ℓ_2 attacks. However this may not necessarily translate to robustness against ℓ_∞ attacks. For this case, one can use an alternate assumption on the distribution of potential adversarial examples.

Assumption 3. *We consider a modification of the distribution in Assumption 2 to account for robustness against ℓ_∞ type attacks:*

$$p(X';X,Y,w,\gamma) = \begin{cases} Z^{-1}_{X,w,\gamma}\exp\left(\mathcal{L}(X';Y,w) - \frac{\gamma}{2}\|X'-X\|_\infty\right) & \text{if} \quad 0 \le \mathcal{L}(X';Y,w) \le R \\ 0 & \text{if} \quad \mathcal{L}(X';Y,w) > R \end{cases} \tag{12}$$

where $\|\cdot\|_\infty$ is the ℓ_∞ norm on the vectorization of its argument and $Z_{X,w,\gamma}$ normalizes the probability.

Now the corresponding *Data Entropy Loss* for ℓ_∞ defense is:

$$\mathcal{L}_{de,\infty}(w;X,Y) = Z^{-1}_{X,w,\gamma}\int_{X'}\mathcal{L}(X';Y,w)\exp\left(\mathcal{L}(X';Y,w) - \frac{\gamma}{2}\|X-X'\|_\infty\right)dX'$$

and our new objective is to minimize this augmented objective function $\mathcal{L}_{de,\infty}(w;X,Y)$, which resembles a averaged version of the loss function with a exponential ℓ_∞ kernel along the data dimension which accounts for points in the ℓ_∞ neighborhood of X which have high loss. The SGD update can be designed as follows:

$$\nabla_w\mathcal{L}_{de,\infty}(w;X,Y) = \nabla_w\mathbb{E}_{X'\sim p(X')}\left[\mathcal{L}(w;X',Y)\right]$$
$$\implies w^+ = w - \eta\nabla_w\mathcal{L}_{de,\infty}(w;X,Y)$$

where the expectation over $p(X')$ is computed by using samples generated via Langevin Dynamics:

$$X'^{t+1} = X'^t + \eta\nabla_{X'}\log p(X'^t) + \sqrt{2\eta}\mathcal{N}(0,\mathbb{I})$$

Plugging in the distribution in Eq.12 the update rule for sampling X':

$$\begin{aligned} X'^{t+1} &= X'^t + \eta\nabla_{X'}\left(L(X'^t;Y,w) - \frac{\gamma}{2}\|X-X'\|_\infty\right) + \sqrt{2\eta}\mathcal{N}(0,\mathbb{I}) \\ &= X' + \eta\nabla_{X'}L(X'^t;Y,w) + \gamma\text{sign}(X_i - X'^t_i)\cdot\mathbf{1} + \sqrt{2\eta}\mathcal{N}(0,\mathbb{I}) \end{aligned} \tag{13}$$

where $i = \arg\max_j |X_j - X'^t_j|$ and j scans all elements of the tensors X, X'^t and $\mathbf{1}_j = \delta_{i,j}$. Note that the third step in the update rule navigates the updates X'^{t+1} to lie in the immediate ℓ_∞ neighborhood of X, which replaces the $\text{sgn}(\nabla\mathcal{L}_x(X;Y,w))$ update in IFGS (refer Appendix C in supplement).

Table 1: Training time comparison in seconds.

	SGD	Entropy SGD	ℓ_2 PGD	ℓ_∞ PGD	ℓ_2 Data Entropy SGD	ℓ_∞ Data Entropy SGD
MNIST	1101	1582	2989	3114	3017	3320
CIFAR10	6840	8321	14218	16985	16860	16745

4 Experiments

In this section we perform experiments on LetNet5 trained on MNIST dataset and VGG-11 on CIFAR10. We also replicate all experiments on denser models such as ResNet18 in Appendix D of supplement. We conduct our studies on six different classic as well as robust training algorithms: SGD, Entropy SGD, ℓ_2 Data Entropy SGD, ℓ_∞ Data Entropy SGD, ℓ_2 PGD training, and ℓ_∞ PGD training to train LeNet5 and VGG-11 respectively, compare results and obtain accuracies for classification with a range of adversarial samples corresponding to different ϵ-values for a given $\Delta_{p,\epsilon}$ defined as $\Delta_{p,\epsilon} = \{\delta : \|\delta\|_p \leq \epsilon\}$. The adversarial samples were generated using PGD [11] and IFGS [27] attacks. Further architecture details are provided in Appendix D.

Generating adversarial samples: Attacks are generated according to a range of ϵ values, by using (i) PGD [11] attack on ℓ_2-ball for values $\epsilon = \{0, 0.33, 0.66, \ldots 6.00\}$ and (ii) FGSM attack [27] on ℓ_∞ ball $\epsilon = \{0, 0.025, 0.05, \ldots 0.5\}$. For CIFAR-10, the range of ϵ values using PGD attack on ℓ_2 ball are in the range $\{0, 0.33, 0.66, \ldots 4.00\}$ and for FGSM attack on ℓ_∞ ball, $\epsilon = \{0, 0.01, 0.02, \ldots 0.1\}$.

Training algorithms: (1) SGD: We trained a LeNet5 for the MNIST dataset, setting batch size of 128, SGD optimizer using PyTorch framework with a learning rate of 0.1. We trained a VGG-11 for the CIFAR10 dataset using the same settings as above. We obtained validation accuracy of 99.8% on MNIST dataset and 98.48% on CIFAR10 dataset. These networks provides a baseline model against a variety of adversarially trained networks. Next, we train both networks using (2) Entropy SGD, with $L = 1$ langevin steps and $\gamma = 10^{-4}$, batch size of 128 and learning rate of 0.1. Then we test out the (3) PGD training trained with (i) ℓ_2 attack ball with $\epsilon = 3$ (ii) ℓ_∞ attack ball with $\epsilon = 0.3$.

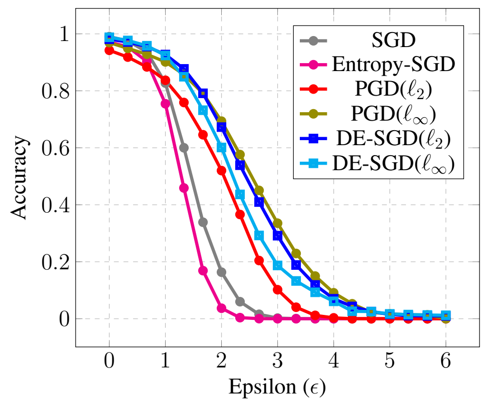
Lastly we test the performance of the algorithms described in this paper, i.e. (3) Data-Entropy SGD designed for (i) ℓ_2 attacks with $L = 20$ langevin steps, $\gamma = 10^{-5}$ and δ_i is a random normally distributed vector with $\|\delta_i\|_2 \leq 3$ for initialization in step 4 of Algorithm 1 (ii) ℓ_∞ attacks with $L = 20$ Langevin steps, $\gamma = 10^{-5}$ with $\|\delta_i\|_\infty \leq 0.3$ for initialization. All algorithms are trained for 50 epochs. The later two broad approaches for robust training also used learning rate of 0.1 and all algorithms were trained for 50 epochs.

MNIST: The results for MNIST dataset were generated using Intel(R) Xeon(R) W-2195 CPU @ 2.30GHz Lambda cluster with 18 cores with NVIDIA TITAN GPU and PyTorch version 1.4.0. We demonstrate our results in Figure 1. As can be seen, for ℓ_2 attack , ℓ_2 Data Entropy SGD trumps ℓ_2 PGD-SGD (Fig.1 (a)) in terms of robustness of classification accuracy against ℓ_2 attack at radius $\ell_2 = 3$. Note that fairness of comparison between ℓ_2 and ℓ_∞ trained approaches largely relies on picking perturbation radii which produce comparable errors in classification.

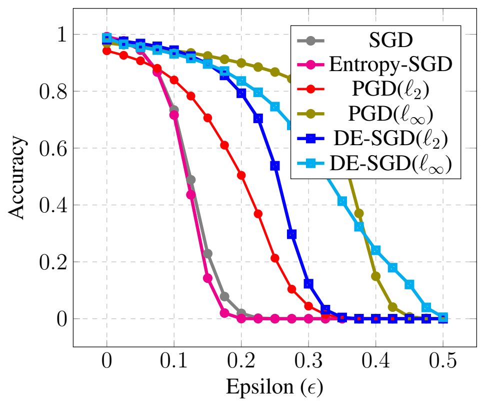
Similarly, Data Entropy SGD (ℓ_∞) trumps PGD-SGD (ℓ_∞) (Fig.1 (b)) at larger attack radii in terms of robustness of classification accuracy against ℓ_∞ attack. In Appendix D Figure 4 we show the class-wise performance of these approaches classwise for digits in MNIST and demonstrate an attack setting at which the models trained with naive SGD and Entropy SGD mispredict the target labels, whereas Data Entropy SGD (DESGD) predicts the correct target labels for those adversarial samples.

CIFAR10: The experiments for CIFAR10 dataset were conducted on Intel(R) Core(TM) i9-7920X CPU @ 2.90GHz lambda cluster with 4 GeForce GTX 1080 Ti GPUs. We test the robustness of all six networks trained on the CIFAR10 dataset. We demonstrate results in Fig. 2. As can be seen, Data Entropy variants outperform its competitors. As can be seen, for ℓ_2 attack , ℓ_2 Data Entropy SGD trumps ℓ_2 PGD-SGD (Fig.2 (a)) in terms of robustness of classification accuracy against ℓ_2 attack. Similarly, ℓ_∞ Data Entropy SGD trumps ℓ_∞ PGD-SGD (Fig.2 (b)) in terms of robustness of classification accuracy against ℓ_∞ attacks.

In Table 1 we tabulate the training times required for all algorithms. As can be seen the running times for Data Entropy SGD are more comparable to the PGD based training approaches. All additional experiments can be found in Appendix D.

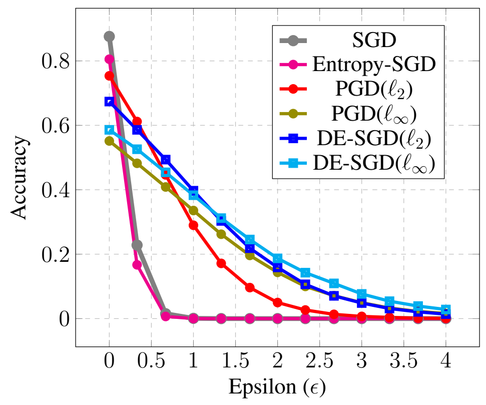


(a) ℓ_2 bounded adversary

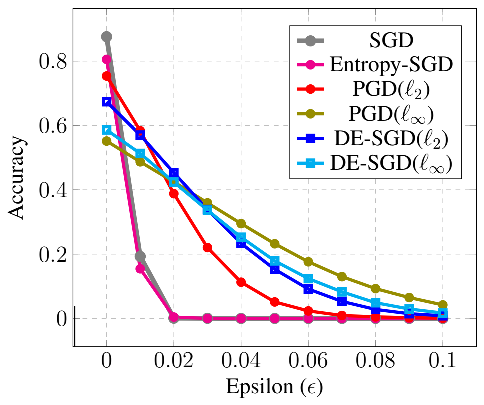


(b) ℓ_∞ bounded adversary

Figure 1: Overall accuracy on MNIST dataset with following networks : 1) SGD 2) entropy-SGD 3) ℓ_2 PGD training 4) ℓ_∞ PGD training 5) DE-SGD(ℓ_2) training 6) DE-SGD(ℓ_∞) training. The networks are attacked by (a) ℓ_2-bound and (b) ℓ_∞ bound adversaries with various ϵ values.



(a) ℓ_2 bounded adversary



(b) ℓ_∞ bounded adversary

Figure 2: Overall accuracy on CIFAR-10 dataset with VGG-11 network variants : 1) SGD 2) entropy-SGD 3) ℓ_2 PGD training 4) ℓ_∞ PGD training 5) DE-SGD(ℓ_2) training 6) DE-SGD(ℓ_∞) training. The networks are attacked by (a) ℓ_2-bound and (b) ℓ_∞ adversaries with various ϵ values.

5 Conclusions and discussion

We propose a new algorithm for robustifying neural networks against adversarial attacks. We demonstrate comparable, and in certain regimes, superlative performance of our algorithm (DESGD) against state of art. Future directions involve a thorough theoretical analysis of our proposed algorithm. We also plan to test out this approach over larger datasets, such as ImageNet.

6 Broader impact

The vulnerability of modern machine learning systems to adversarial attacks is a major impediment to their widespread deployment in safety-critical real world applications (such as self-driving cars). Therefore, it is imperative to develop increasingly more effective and robust machine learning models. Our approach in this paper constitutes an addition to the growing body of work in this literature that addresses this imperative.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[2] W. Zhang, Q. Sheng, A. Alhazmi, and C. Li. Adversarial attacks on deep-learning models in natural language processing: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 11(3):1–41, 2020.

[3] N. Carlini, P. Mishra, T. Vaidya, Y. Zhang, M. Sherr, C. Shields, D. Wagner, and W. Zhou. Hidden voice commands. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pages 513–530, 2016.

[4] J. Tang, X. Du, X. He, F. Yuan, Q. Tian, and T. Chua. Adversarial training towards robust multimedia recommender system. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[5] P. Rota, E. Sangineto, V. Conotter, and C. Pramerdorfer. Bad teacher or unruly student: Can deep learning say something in image forensics analysis? In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2503–2508. IEEE, 2016.

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

[7] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[8] B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[9] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[10] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018.

[12] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples, 2018.

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

[14] J. Cohen, E. Rosenfeld, and Z. Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1310–1320, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[15] Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In *Advances in Neural Information Processing Systems*, pages 11289–11300, 2019.

[16] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

[17] M. Welling and Y. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 681–688, 2011.

[18] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training, 2020.

[19] K. Ren, T. Zheng, Z. Qin, and X. Liu. Adversarial attacks and defenses in deep learning. *Engineering*, 2020.

[20] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[21] Sumanth Dathathri, Stephan Zheng, Sicun Gao, and RM Murray. Measuring the Robustness of Neural Networks via Minimal Adversarial Examples. In *NeurIPS-W*, volume 35, 2017.

[22] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[23] Ian J. Goodfellow. Defense against the dark arts: An overview of adversarial example security research and future research directions. *arxiv preprint*, abs/1806.04169, 2018.

[24] 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.

[25] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *CVPR*, 2017.

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

[27] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[28] Y. Dong, F. Liao, T. Pang, H Su, J. Zhu, X. Hu, and J. Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[29] Ali Shafahi, Mahyar Najibi, Mohammad Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Processing Systems*, pages 3353–3364, 2019.

[30] Bai Li, Changyou Chen, Wenlin Wang, and Lawrence Carin. Certified adversarial robustness with additive gaussian noise. *arXiv preprint arXiv:1809.03113*, 2018.

[31] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[32] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[33] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

A Recap: Entropy SGD

Algorithm 2 Entropy SGD

```
 1: Input: $X = [X_{B_1}, X_{B_2} \dots X_{B_J}], f, \eta, \eta', w = w^0, \gamma, \alpha, \varepsilon$
 2: for $t = 0, \cdots T-1$ do
 3:     for $j = 1, \cdots J$ do
 4:         $w'^0 \leftarrow w^t, \mu^0 \leftarrow w^t$          {Repeat inner loop for all training batches $j$}
 5:         for $k = 0, \cdots, K-1$ do
 6:             $dw'^k \leftarrow \frac{1}{n_j}\sum_{i=1}^{n_j} -\nabla_{w=w^k} L(f(w;x_i)) + \gamma(w^k - w'^k)$ {$\forall x_i \in X_{B_j}$}
 7:             $w'^{k+1} \leftarrow w'^k + \eta' dw'^k + \sqrt{2\eta'}\varepsilon\mathcal{N}(0,1)$ {Langevin update}
 8:             $\mu^k \leftarrow (1-\alpha)\mu^k + \alpha w'^{k+1}$
 9:         end for
10:         $\mu^t \leftarrow \mu^K$
11:         $w^{t+1} \leftarrow w^t - \eta\gamma(w^t - \mu^t)$ {Repeat outer loop step for all training batches $j$}
12:     end for
13: end for
14: Output $\hat{w} \leftarrow w^T$
```

In [16] authors claim that neural networks that favor wide local minima have better generalization properties, in terms of perturbations to data, weights as well as activations. Mathematically, the formulation in Entropy SGD can be summarized as follows. A basic way to model the distribution of the weights of the neural network is using a Gibbs distribution of the form:

$$p(w; X, Y, \beta) = Z_{X,\beta}^{-1} \exp\left(-\beta\mathcal{L}(w; X, Y)\right)$$

When $\beta \to \infty$, this distribution concentrates at the global (if unique) minimizer of $\mathcal{L}(w^*; X, Y)$. A modified Gibbs distribution, with an additional smoothing parameter is introduced, which assumes the form:

$$p(w'; w, X, Y, \beta = 1, \gamma) = Z_{w,X,\gamma}^{-1} \exp\left(-\mathcal{L}(w'; X, Y) - \frac{\gamma}{2}\|w' - w\|_2^2\right) \tag{14}$$

where $Z_{w,X,\gamma}$ normalizes the probability.

Here γ controls the width of the valley; if $\gamma \to \infty$, the sampling is sharp, and this corresponds to no smoothing effect, meanwhile $\gamma \to 0$ corresponds to a uniform contribution from all points in the loss manifold. The standard objective is:

$$\begin{aligned}\min_w \mathcal{L}(w; X, Y) &:= \min_w -\log\Big(\exp\left(-\mathcal{L}(w; X, Y)\right)\Big) \\ &= \min_w -\log\left(\int_{w'} \exp\Big(-\mathcal{L}(w'; X, Y)\Big)\delta(w - w')dw'\right)\end{aligned}$$

which can be seen as a sharp sampling of the loss function. Now, if one defined the Local Entropy as:

$$\begin{aligned}\mathcal{L}_{ent}(w; X, Y) &= -\log(Z_{w,X,Y,\gamma}) \\ &= -\log\left(\int_{w'} \exp\left(-\mathcal{L}(w'; X, Y) - \frac{\gamma}{2}\|w - w'\|_2^2\right)dw'\right)\end{aligned}$$

our new objective is to minimize this augmented objective function $\mathcal{L}_{ent}(w; X, Y)$, which resembles a smoothed version of the loss function with a Gaussian kernel. The SGD update can be designed as follows:

$$\begin{aligned}\nabla_w\mathcal{L}_{ent}(w; X, Y) &= -\nabla_w(\log(Z_{w,X,Y,\gamma})) \\ &= Z_{w,X,\gamma}^{-1}\nabla_w(Z_{w,X,\gamma}) \\ &= Z_{w,X,\gamma}^{-1}\left(\int_{w'} \exp\left(-\mathcal{L}(w'; X, Y) - \frac{\gamma}{2}\|w - w'\|_2^2\right) \cdot \gamma(w - w')dw'\right) \\ &= \int_{w'} p(w'; w, X, Y, \gamma) \cdot \gamma(w - w')dw' \\ &= \mathbb{E}_{w' \sim p(w')}\Big[\gamma(w - w')\Big]\end{aligned}$$

Then, using this gradient, the SGD update for a given batch is designed as:

$$w^+ = w - \eta\nabla_w\mathcal{L}_{ent}(w; X, Y)$$

Algorithm 3 PGD SGD

```
1: Input: $[X_{B_1}, X_{B_2} \dots X_{B_J}], f, \eta, \eta', w = w^0, \epsilon$
2: for $t = 0, \cdots T-1$ do
3:     for $j = 1, \cdots J$ do
4:         $x'^0 \leftarrow x$ $\{\forall x \in X_{B_j}\}$
5:         for $k = 0, \cdots, K-1$ do
6:             $dx'^k \leftarrow \frac{1}{n}\sum_{i=1}^{n} \nabla_{x=x^k} L(f(w^t; x))$
7:             $x'^{k+1} \leftarrow x'^k + \eta' dx'^k$ {Gradient ascent}
8:             project$_{x+\Delta}(x', \epsilon)$
9:         end for
10:        $\mu^t \leftarrow L(w^t, x'^K)$ {batch loss for $X_{B_j}$}
11:        $dL^t \leftarrow \nabla_w \mu^t$ {gradient of batch loss}
12:        $w^{t+1} \leftarrow w^t - \eta dL^t$
13:    end for
14: end for
15: Output $\hat{w} \leftarrow w^T$
```

This gradient ideally requires computation over the entire training set at once; however can be extended to a batch-wise update rule by borrowing key findings from [17]. This expectation for the full gradient is computationally intractable, however, Euler discretization of Langevin Stochastic Differential Equation, it can be approximated fairly well as

$$w'^{t+1} = w'^t + \eta^t \nabla_{w'} \log p(w'^t) + \sqrt{2\eta}\mathcal{N}(0, \mathbb{I})$$

such that after large enough amount of iterations $w^+ \to w^\infty$ then $w^\infty \sim p(w')$. One can estimate $\mathbb{E}_{w' \sim p(w')}\left[\gamma(w - w')\right]$ by averaging over many such iterates from this process. This result is stated as it is from [16]: $\mathbb{E}_{w' \sim p(w')}[g(w')] = \frac{\sum_t \eta_t g(w'_t)}{\sum_t \eta_t}$. This leads to the algorithm shown in Algorithm 2. One can further accrue exponentially decaying weighted averaging of $g(w'_t)$ to estimate $\mathbb{E}_{w' \sim p(w')}[g(w')]$. This entire procedure is described in Algorithm 2.

This algorithm is then further guaranteed to find wide minima neighborhoods of w by design, as sketched out by the proofs in [16].

B Stochastic Gradient Langevin Dynamics

Stochastic Gradient Langevin Dynamics combines techniques of Stochastic Gradient Descent and Langevin Dynamics [17]. Given a probability distribution $\pi = p(\theta; X)$, the following update rule allows us to sample from the distribution π:

$$\theta^{t+1} = \theta^t + \eta \nabla_\theta p(\theta^t; X) + 2\sqrt{\eta'}\varepsilon \tag{15}$$

where η' is step size and ε is normally distributed. Then, as $t \to \infty$, $\theta \sim \pi$.

While this update rule in itself suffices, if the parameters are conditioned on a a training sample set X, which is typically large, the gradient term in Eq. 15 is expensive to compute. [17] shows that the following batch-wise update rule:

$$\theta^{t+1} = \theta^t + \eta \nabla_\theta p(\theta^t; X_{B_j}) + 2\sqrt{\eta'}\varepsilon$$

suffices to produce a good approximation to the samples $\theta^{t \to \infty} \sim \pi$. In Algorithm 1, θ is the set of perturbed points X'. In each internal iteration, we look at subset of trainable parameters X'_{B_j}. We update the estimate for X'_{B_j} by only considering data-points X_{B_j} at a time. In the current formulation the set of iterable parameters X'_{B_j} only 'see' a single batch of data X_{B_j} ; a better estimate would require X'_{B_j} to be updated by iteratively over all possible batches X_{B_k}, $k = 1, 2 \dots J$. However in practice we observe that just using the corresponding batch $X_{B_{k=j}}$ suffices. In future work, we will explore the theoretical implications of this algorithmic design.

C Universal defense: PGD-SGD and IFGS

In Algorithm 3 we describe the PGD-SGD algorithm. In [11] authors demonstrate that PGD based-attack is the best possible attack that can be given for a given network and dataset combination. Theoretically,

$$\bar{x}_{\text{worst}} = \arg\max_{\delta \in \Delta_p} L(f(\hat{w}; x + \delta), y) \tag{16}$$

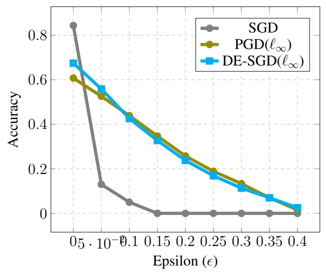


Figure 3: Test accuracy under ℓ_∞ adversarial attack on ResNet18 models trained on CIFAR10.

and if this maximization can be solved tractably, then a network trained with the following min-max formulation is said to be robust:

$$\min_{w} \max_{\delta \in \Delta_p} \mathcal{L}(f(\hat{w}; X + \delta, Y), Y) \tag{17}$$

Furthermore they show that first order based gradient approaches, such as SGD, are sufficient to suitably optimize the inner maximization over the perturbed dataset. This can be obtained using the following gradient *ascent* update rule:

$$X' = X' + \eta' \nabla_{X' \in X+\delta} \mathcal{L}(f(w, X' + \delta; Y), Y)$$

(see also Step 7 of Algorithm 3). Note that when $\Delta_p = \Delta_2$, this projection rule represents a noise-less version of the update rule in DESGD (see Algorithm 1, line 8).

Iterative Fast Gradient Sign (IFGS) method effectively captures a similar projection based approach which performs an update within an ℓ_∞ ball. This update is given by:

$$X' = X' + \eta' \text{sign}(\nabla_{X'} \mathcal{L}(f(w, X' + \delta; Y), Y))$$

Note that this update rule constructs an adversarial example within ℓ_∞-ball, during the training procedure.[2] Meanwhile, given our proposed adversarial example sampling criterion in Assumption 3, our update rule is slightly different (see also Eq. 13).

D Additional experiments and details

We provide additional details as well as experiments to supplement those in Section 4. We run all experiments on the MNIST dataset using the LeNet5 network configuration. This network consists of three convolutional layers with two max-pooling layers interspersed, along with a final fully connected layer. The activations after each convolution are followed with a ReLU activation. The output of the final fully connected layer is followed by a sigmoid (softmax) activation.

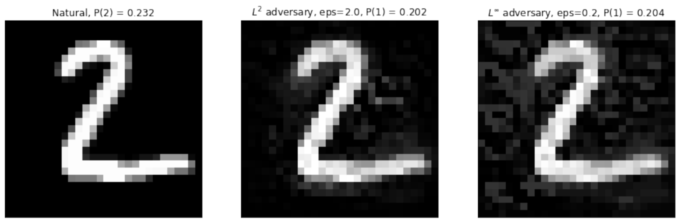
In Figure 4 we show the predictions of various different networks on adversarial examples generated on MNIST. We can see that SGD and Entropy SGD trained networks are incapable of producing the correct label for this example (they both predict “1" for true label “2"), where as PGD-retrained as well as DESGD produces the correct label, which is “2", even under significant data corruptions.

The experiments on CIFAR10 datasets are done using two different networks. Experiments on VGG-11 are shown in the main paper (Section 4). We also show additional experiments on ResNet18 network below. The configuration of VGG-11 is as follows. This architecture consists of eight convolutional layers, five maxpooling operations, interspersed with ReLU activation. This is followed by three fully-connected layers and softmax operation. Similarly ResNet18 consists of 18 layers and 5 residual blocks.

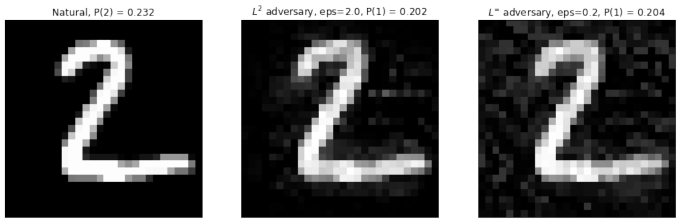
We provide experimental results on ResNet18 trained on CIFAR10 dataset in Figure 3. We compare between three approaches (i) standard network trained using SGD, (ii) adversarially trained network using FGSM, (iii) the ℓ_∞ formulation of DESGD, using update rule Eq. 13. We demonstrate that out proposed approach, generalizes well even on ResNet type configurations. For the ResNet experiment, we use the following settings for each algorithm: (i) SGD, learning rate=0.05, momentum=0.9, 50 epochs (ii) PGD-SGD (ℓ_∞), learning rate=0.1, 20 epochs and $\varepsilon = 0.2$, 10 inner iterations (iii) DESGD (ℓ_∞) with step size 0.01, learning rate-0.1, initial point δ, such that $\|X'^0_{B_j} - X_{B_j}\|_\infty = 0.2$, 10 inner iterations with step size 0.01, $\gamma = 10^{-7}$. Note that the performance of all networks may be further improved by doing a better hyper-parameter search.

For the attack, we used used range of perturbations $\varepsilon_\infty = [0, 0.4]$, with a step of 0.05. The attacks were generated using IFGS method.

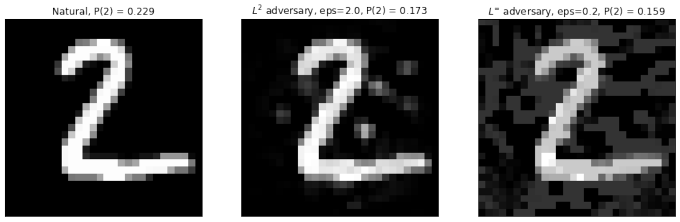
[2]Note that we refer to IFGS as PGD-SGD(ℓ_∞), interchangeably in this paper.


Natural, P(2) = 0.232
L^2 adversary, eps=2.0, P(1) = 0.202
L^∞ adversary, eps=0.2, P(1) = 0.204

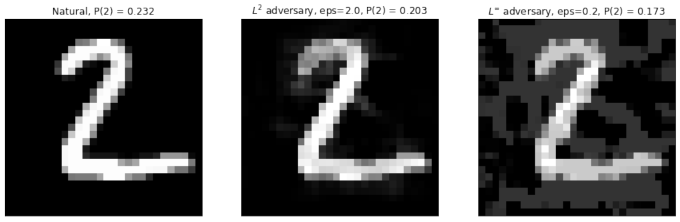
(a) SGD model


Natural, P(2) = 0.232
L^2 adversary, eps=2.0, P(1) = 0.202
L^∞ adversary, eps=0.2, P(1) = 0.204

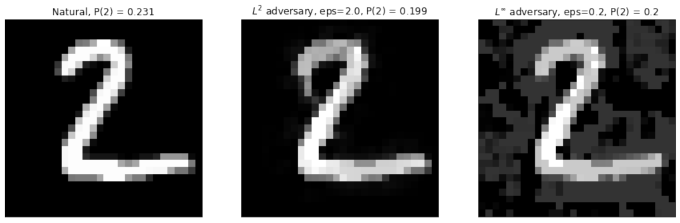
(b) Entropy SGD model


Natural, P(2) = 0.229
L^2 adversary, eps=2.0, P(2) = 0.173
L^∞ adversary, eps=0.2, P(2) = 0.159

(c) Data Entropy SGD ℓ_2 model


Natural, P(2) = 0.232
L^2 adversary, eps=2.0, P(2) = 0.203
L^∞ adversary, eps=0.2, P(2) = 0.173

(d) PGD ℓ_2-retrained


Natural, P(2) = 0.231
L^2 adversary, eps=2.0, P(2) = 0.199
L^∞ adversary, eps=0.2, P(2) = 0.2

(e) PGD ℓ_∞- retrained

Figure 4: [MNIST] SGD, Entropy-SGD, Data Entropy-SGD (ℓ_2 model), PGD $\ell_2$retrained and PGD ℓ_∞ retrained model performance on clean and adversarial samples. We display the class-probability corresponding to the class which has highest probability. The examples in (a) and (b) are misclassified, however (c)-(e) are correctly classified.