Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

April 8, 2023

Today's Agenda

Recap

Intriguing properties of neural networks

Intriguing properties of neural networks

Christian Szegedy
Google Inc.

Wojciech Zaremba
New York University

Ilya Sutskever
Google Inc.

Joan Bruna
New York University

Dumitru Erhan
Google Inc.

Ian Goodfellow
University of Montreal

Rob Fergus
New York University
Facebook Inc.

Adversarial Examples

The authors found that applying an **imperceptible non-random perturbation** to a test image, it is possible to arbitrarily **change the network's prediction**.

- They term the so perturbed examples **Adversarial Examples**.
- They found that adversarial examples are relatively robust, and are **shared by neural networks** with varied number of layers, activations or trained on different subsets of the training data (**Transferability**).

Smoothness Prior

For a small enough radius $\epsilon \geq 0$ in the vicinity of a given training input x, an $x + r$ satisfying $\|r\| \leq \epsilon$ will get assigned a high probability of the correct class by the model.

- This kind of smoothness prior is typically valid for computer vision problems.
- In general, imperceptibly tiny perturbations of a given image do not normally change the underlying class.

The main result of the paper is that **for deep neural networks, the smoothness assumption does not hold**.

Blind Spots

- In some sense, what we describe is a way to **traverse the manifold** represented by the network in an efficient way (by optimization) and finding **adversarial examples** in the input space.
- The adversarial examples represent **low-probability (high-dimensional) “pockets” in the manifold,** which are hard to efficiently find by simply randomly sampling the input around a given example.

Formal description

For a given $x \in \mathbb{R}^m$ image and target label $l \in \{1...k\}$, we aim to solve the following box-constrained optimization problem:

$$\text{Minimize } \|r\|_2 \text{ subject to:}$$
$$f(x+r) = l$$
$$x + r \in [0,1]^m$$

- Informally, $x' = x + r$ is the closest image to x classified as l by f.

Formal description

For a given $x \in \mathbb{R}^m$ image and target label $l \in \{1...k\}$, we aim to solve the following box-constrained optimization problem:

$$\text{Minimize } \|r\|_2 \text{ subject to:}$$
$$f(x+r) = l$$
$$x + r \in [0,1]^m$$

- Informally, $x' = x + r$ is the closest image to x classified as l by f.
- The minimizer r might not be unique.
- This task is non-trivial only if $f(x) \neq l$.
- In general, the exact computation of x' is a hard problem, so we approximate it by using a box-constrained L-BFGS.

Formal description

Recall: Generalized Lagrange Function (Karush–Kuhn–Tucker (KKT))

Suppose we wish to maximize $f(x)$ subject to $g_j(x) = 0$ for $j = 1, \cdots, J$, and $h_k(x) \geq 0$ for $k = 1, \cdots, K$.

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & g_j(x) = 0 \quad \text{for} \quad j = 1, \cdots, J \\ & h_k(x) \geq 0 \quad \text{for} \quad k = 1, \cdots, K \end{aligned}$$

We introduce Lagrange multipliers $\{\lambda_j\}$ and $\{\mu_k\}$, and then optimize the Lagrangian function given by

$$L(x, \{\lambda_j\}, \{\mu_k\}) = f(x) + \sum_{j=1}^{J} \lambda_j g_j(x) + \sum_{k=1}^{K} \mu_k h_k(x)$$

subject to $\mu_k \geq 0$ and $\mu_k h_k(x) = 0$ for $k = 1, \cdots, K$.

The optimal point $x^\star$ of the above constrained optimization on $f(x)$ is the same as the optimal point of the unconstrained optimization L.

(See this playlist for more information about Lagrange multipliers)

Formal description

For a given $x \in \mathbb{R}^m$ image and target label $l \in \{1...k\}$, we aim to solve the following box-constrained optimization problem:

$$\text{Minimize } \|r\|_2 \text{ subject to:}$$
$$f(x+r) = l$$
$$x + r \in [0,1]^m$$

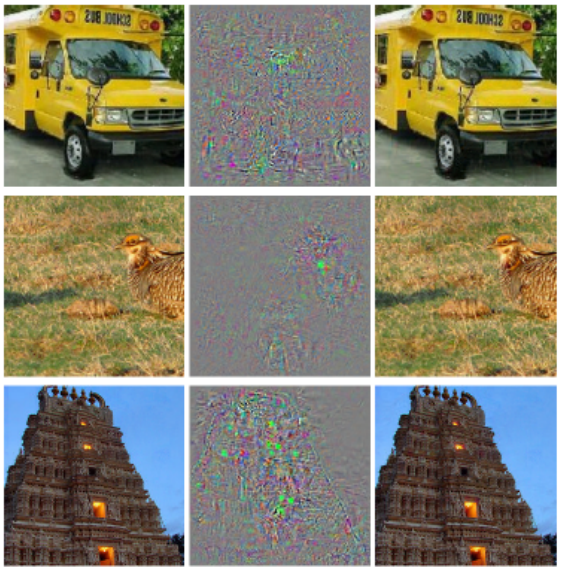
- Informally, $x' = x + r$ is the closest image to x classified as l by f.
- The minimizer r might not be unique.
- This task is non-trivial only if $f(x) \neq l$.
- In general, the exact computation of x' is a hard problem, so we approximate it by using a box-constrained L-BFGS.

Concretely, we find an approximation of x' by performing line-search to find the minimum $c > 0$ for which the minimizer r of the following problem satisfies $f(x+r) = l$.

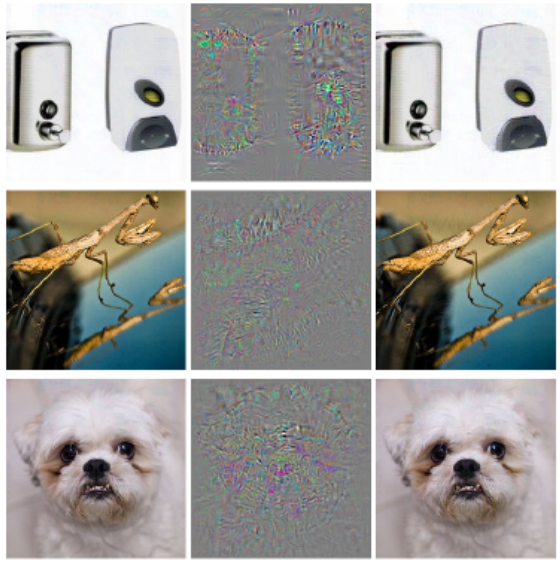
$$\text{Minimize } c\|r\|_2 + loss_f(x+r, l) \text{ subject to } x + r \in [0,1]^m$$

- Since neural networks are non-convex in general, so we end up with an approximation to find solution.

Adversarial Examples



(a)



(b)

Figure 5: Adversarial examples generated for AlexNet [9].(Left) is a correctly predicted sample, (center) difference between correct image, and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial example. All images in the right column are predicted to be an *"ostrich, Struthio camelus"*. Average distortion based on 64 examples is 0.006508. Plase refer to `http://goo.gl/huaGPb` for full resolution images. The examples are strictly randomly chosen. There is not any postselection involved.

Experimental results

Intriguing properties

- **100%** success rate
  - For all the networks we studied (MNIST, AlexNet (ImageNet)), for each sample, we have always managed to generate very close, visually hard to distinguish, adversarial examples that are misclassified by the original network.

Experimental results

Intriguing properties

- **100%** success rate
  - For all the networks we studied (MNIST, AlexNet (ImageNet)), for each sample, we have always managed to generate very close, visually hard to distinguish, adversarial examples that are misclassified by the original network.
- **Cross model generalization**
  - A relatively large fraction of examples will be misclassified by networks trained from scratch with **different hyper-parameters** (number of layers, regularization or initial weights).

Experimental results

Intriguing properties

- **100%** success rate
  - For all the networks we studied (MNIST, AlexNet (ImageNet)), for each sample, we have always managed to generate very close, visually hard to distinguish, adversarial examples that are misclassified by the original network.
- **Cross model generalization**
  - A relatively large fraction of examples will be misclassified by networks trained from scratch with **different hyper-parameters** (number of layers, regularization or initial weights).
- **Cross training-set generalization**
  - A relatively large fraction of examples will be misclassified by networks trained from scratch on a **disjoint training set**.

Experimental results

Intriguing properties

- **100%** success rate
  - For all the networks we studied (MNIST, AlexNet (ImageNet)), for each sample, we have always managed to generate very close, visually hard to distinguish, adversarial examples that are misclassified by the original network.
- **Cross model generalization**
  - A relatively large fraction of examples will be misclassified by networks trained from scratch with **different hyper-parameters** (number of layers, regularization or initial weights).
- **Cross training-set generalization**
  - A relatively large fraction of examples will be misclassified by networks trained from scratch on a **disjoint training set**.

The above observations suggest that adversarial examples are somewhat **universal** and not just the results of overfitting to a particular model or to the specific selection of the training set.

Intriguing properties of neural networks

Spectral Analysis of Unstability

- The adversarial examples show that there exist **small additive perturbations** of the input (in Euclidean sense) that produce **large perturbations at the output** of the last layer.
- Mathematically, if $\phi(x)$ denotes the output of a network of K layers corresponding to input x and trained parameters W, we write

$$\phi(x) = \phi_K(\phi_{K-1}(...\phi_1(x; W_1)...; W_{K-1})W_K)$$

where ϕ_K denotes the operator mapping layer $k-1$ to layer k.

Spectral Analysis of Unstability

- The adversarial examples show that there exist **small additive perturbations** of the input (in Euclidean sense) that produce **large perturbations at the output** of the last layer.
- Mathematically, if $\phi(x)$ denotes the output of a network of K layers corresponding to input x and trained parameters W, we write

$$\phi(x) = \phi_K(\phi_{K-1}(...\phi_1(x; W_1)...; W_{K-1})W_K)$$

  where ϕ_K denotes the operator mapping layer $k-1$ to layer k.
- The unstability of $\phi(x)$ can be explained by inspecting the upper **Lipschitz constant** of each layer.

Lipschitz continuity

A function $f : I \rightarrow R$ over some set $I \subseteq \mathbb{R}^d$ is called Lipschitz continuous if there exists a positive real constant L such that, for all $x, y \in I$,

$$|f(y) - f(x)| \leq L\|y - x\|_2$$

or

$$f(x) - L\|y - x\|_2 \leq f(y) \leq f(x) + L\|y - x\|_2$$

We call L the Lipschitz constant of f over I.

Lipschitz continuity

A function $f : I \to R$ over some set $I \subseteq \mathbb{R}^d$ is called Lipschitz continuous if there exists a positive real constant L such that, for all $x, y \in I$,

$$|f(y) - f(x)| \leq L\|y - x\|_2$$

or

$$f(x) - L\|y - x\|_2 \leq f(y) \leq f(x) + L\|y - x\|_2$$

We call L the Lipschitz constant of f over I.

Let functions f_1 and f_2 be both Lipschitz continuous with constants L_1 and L_2, the upper Lipschitz constant of their composition $f_1 \circ f_2$ is $L_1 L_2$.

Lipschitz continuity

A function $f : I \to R$ over some set $I \subseteq \mathbb{R}^d$ is called Lipschitz continuous if there exists a positive real constant L such that, for all $x, y \in I$,

$$|f(y) - f(x)| \leq L\|y - x\|_2$$

or

$$f(x) - L\|y - x\|_2 \leq f(y) \leq f(x) + L\|y - x\|_2$$

We call L the Lipschitz constant of f over I.

Let functions f_1 and f_2 be both Lipschitz continuous with constants L_1 and L_2, the upper Lipschitz constant of their composition $f_1 \circ f_2$ is $L_1 L_2$.

$$|f_1(f_2(y)) - f_1(f_2(x))| \leq L_1|f_2(y) - f_2(x)| \leq L_1 L_2\|y - x\|_2$$

Lipschitz continuity

A function $f : I \to R$ over some set $I \subseteq \mathbb{R}^d$ is called Lipschitz continuous if there exists a positive real constant L such that, for all $x, y \in I$,

$$|f(y) - f(x)| \leq L\|y - x\|_2$$

or

$$f(x) - L\|y - x\|_2 \leq f(y) \leq f(x) + L\|y - x\|_2$$

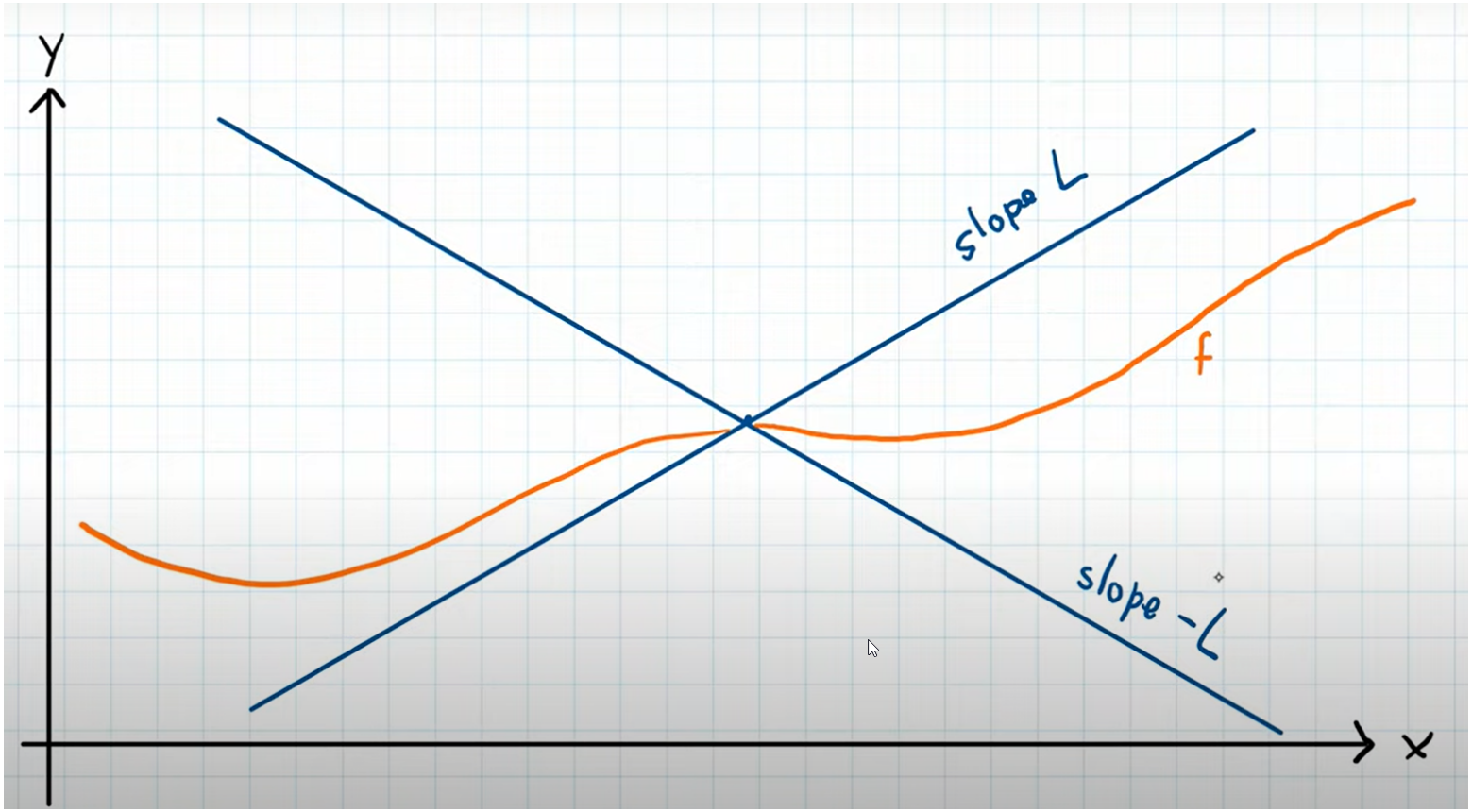
We call L the Lipschitz constant of f over I.

Let functions f_1 and f_2 be both Lipschitz continuous with constants L_1 and L_2, the upper Lipschitz constant of their composition $f_1 \circ f_2$ is $L_1 L_2$.

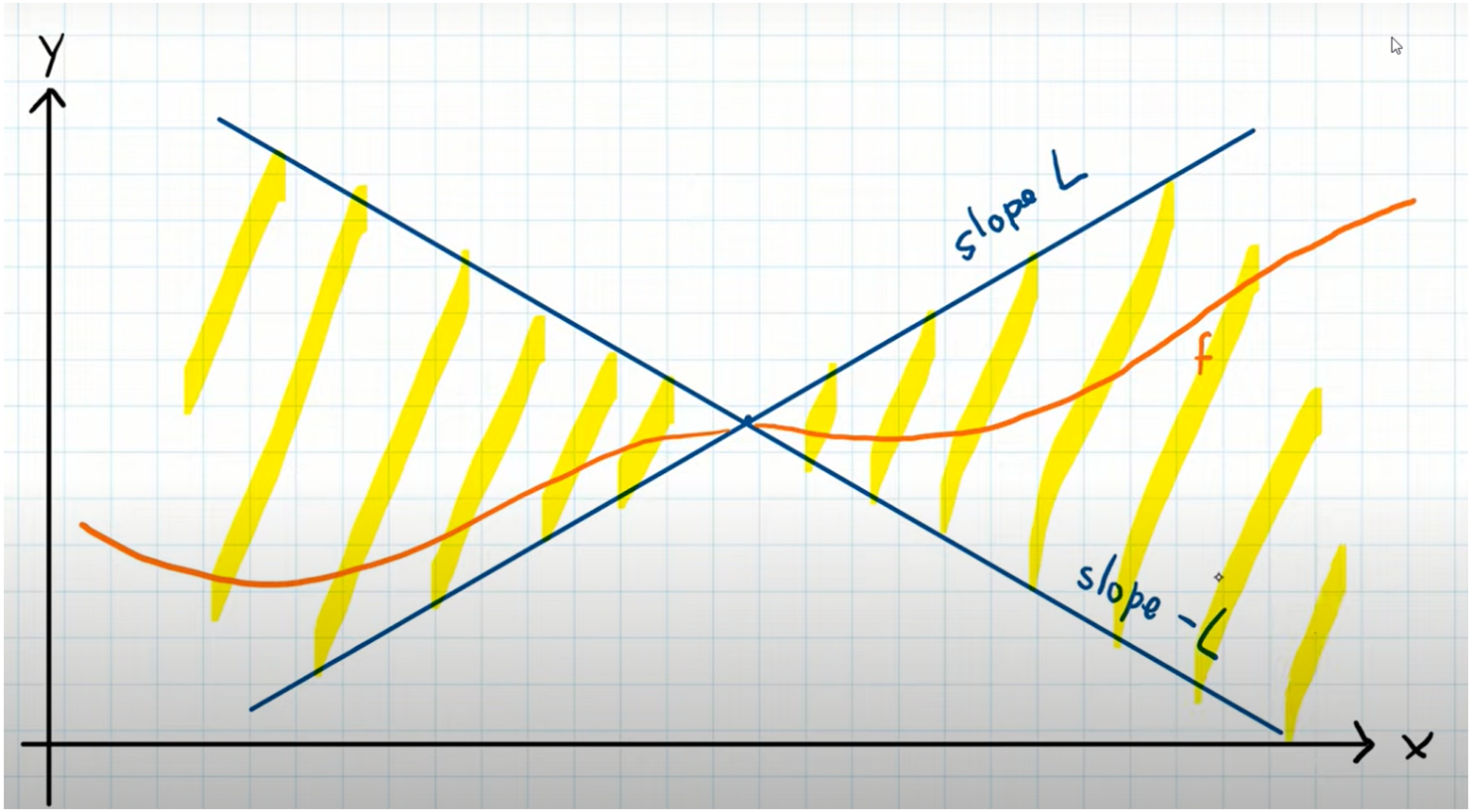
$$|f_1(f_2(y)) - f_1(f_2(x))| \leq L_1|f_2(y) - f_2(x)| \leq L_1 L_2\|y - x\|_2$$

Generally, Let $f = f_1 \circ f_2 \circ .... \circ f_K$ and the Lipschitz constant of f_i be L_i for all $i \in \{1, 2, ..., K\}$, then the Lipschitz constant of f is $L \leq \prod_{k=1}^{K} L_k$.

Lipschitz continuity



Lipschitz continuity



Spectral Analysis of Unstability

- Mathematically, if $\phi(x)$ denotes the output of a network of K layers corresponding to input x and trained parameters W, we write

$$\phi(x) = \phi_K(\phi_{K-1}(...\phi_1(x; W_1)...; W_{K-1})W_K)$$

  where ϕ_K denotes the operator mapping layer $k-1$ to layer k.
- The unstability of $\phi(x)$ can be explained by inspecting the *upper Lipschitz constant* of each layer, defined as the constant $L_k > 0$ such that

$$\forall x, r, \ \|\phi_k(x; W_k) - \phi_k(x + r; W_k)\| \leq L_k \|r\|$$

- The resulting network thus satsifies $\|\phi(x + r) - \phi(x)\| \leq L\|r\|$, with $L \leq \prod_{k=1}^{K} L_k$.

Lipschitz continuity

Let $f : I \rightarrow R$ be a continuous and differntiable function over some set $I \subseteq \mathbb{R}^d$, if we have $\|f'(x)\|_2 \leq m$ for all $x \in I$, then m is the upper Lipschitz constant of f ($L \leq m$).

Lipschitz continuity

Let $f : I \to R$ be a continuous and differntiable function over some set $I \subseteq \mathbb{R}^d$, if we have $\|f'(x)\|_2 \leq m$ for all $x \in I$, then m is the upper Lipschitz constant of f ($L \leq m$).

Proof sketch:
Mean value theorem: Let $f : I \to R$ be a continuous and differntiable function over some set $I \subseteq \mathbb{R}^d$, For all $a, b \in I$ $(b > a)$, there exists some $c \in (a, b)$ such that:

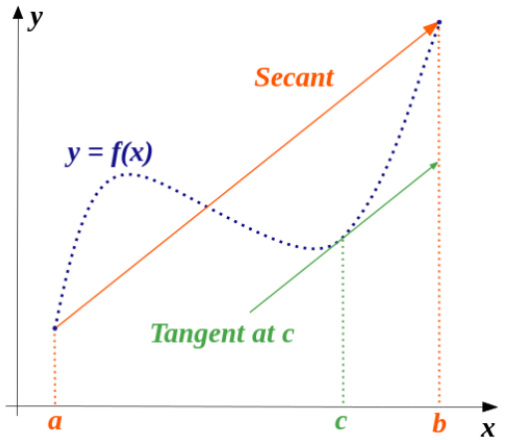
$$f'(c) = \frac{f(b) - f(a)}{b - a}$$

For all $a, b \in I$, there exist $c \in (a, b)$, such that:

$$|f(b) - f(a)| = \|f'(c).b - a\|_2 \leq \|f'(c)\|_2 \|b - a\|_2.$$

Hence, For all $a, b \in I$, if $\|f'(z)\|_2 \leq m$ for $z \in I$, then

$$|f(b) - f(a)| \leq m\|b - a\|_2.$$



Spectral Analysis of Unstability

- ReLU and max pooling layers have a Lipschitz constant of 1.
  - The upper bound of derivative is 1.

Spectral Analysis of Unstability

- ReLU and max pooling layers have a Lipschitz constant of 1.
  - The upper bound of derivative is 1.
- Batch normalization layer has a Lipschitz constant of $\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
  - $\nabla_x BN(x) = \nabla_x \gamma \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$

Spectral Analysis of Unstability

- ReLU and max pooling layers have a Lipschitz constant of 1.
  - The upper bound of derivative is 1.
- Batch normalization layer has a Lipschitz constant of $\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
  - $\nabla_x BN(x) = \nabla_x \gamma \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
- Linear layers (Wx) have the Lipschitz constant of $\sigma(W)$, where σ is the spectral norm (largest singular value).
  - Lipschits constant of linear layers

$$\|Wy - Wx\|_2 \leq L\|y - x\|_2 \Rightarrow \|W(y-x)\|_2 \leq L\|y-x\|_2$$

$$\underset{z=y-x}{\Rightarrow} \|Wz\|_2 \leq L\|z\|_2 \Rightarrow L \geq \frac{\|Wz\|_2}{\|z\|_2} \Rightarrow L = \sigma(W)$$

Spectral Analysis of Unstability

- ReLU and max pooling layers have a Lipschitz constant of 1.
  - The upper bound of derivative is 1.
- Batch normalization layer has a Lipschitz constant of $\frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
  - $\nabla_x BN(x) = \nabla_x \gamma \frac{x-\mu}{\sqrt{\sigma^2+\epsilon}} + \beta = \frac{\gamma}{\sqrt{\sigma^2+\epsilon}}$
- Linear layers (Wx) have the Lipschitz constant of $\sigma(W)$, where σ is the spectral norm (largest singular value).
  - Lipschits constant of linear layers

$$\|Wy - Wx\|_2 \leq L\|y-x\|_2 \Rightarrow \|W(y-x)\|_2 \leq L\|y-x\|_2$$

$$\underset{z=y-x}{\Rightarrow} \|Wz\|_2 \leq L\|z\|_2 \Rightarrow L \geq \frac{\|Wz\|_2}{\|z\|_2} \Rightarrow L = \sigma(W)$$

  - The spectral norm of a matrix $A \in \mathbb{R}^{m\times n}$ is defined as

$$\sigma(A) = \max_{x\in\mathbb{R}^n, x\neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

    which corresponds to the largest singular value of A

Spectral norm definition source.

Spectral Analysis of Unstability

Layer	Size	Stride	Upper bound
Conv. 1	$3 \times 11 \times 11 \times 96$	4	2.75
Conv. 2	$96 \times 5 \times 5 \times 256$	1	10
Conv. 3	$256 \times 3 \times 3 \times 384$	1	7
Conv. 4	$384 \times 3 \times 3 \times 384$	1	7.5
Conv. 5	$384 \times 3 \times 3 \times 256$	1	11
FC. 1	9216×4096	N/A	3.12
FC. 2	4096×4096	N/A	4
FC. 3	4096×1000	N/A	4

Table 5: Frame Bounds of each rectified layer of the network from [9].

$$2.75 \times 10 \times 7 \times 7.5 \times 11 \times 3.12 \times 4 \times 4 \approx 793000$$

- Notice that we compute upper bounds: large bounds do not automatically translate into existence of adversarial examples; however, small bounds guarantee that no such examples can appear.

Explaining and Harnessing Adversarial Examples

Explaining and Harnessing Adversarial Examples

Published as a conference paper at ICLR 2015

EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES

Ian J. Goodfellow, Jonathon Shlens & Christian Szegedy
Google Inc., Mountain View, CA
{goodfellow,shlens,szegedy}@google.com

Abstract

- We argue the primary cause of neural networks' vulnerability to adversarial perturbation is their **linear nature**.
- Giving the first explanation of the most intriguing fact about them: their **generalization** across architectures and training sets.
- We propose a **simple and fast method of generating adversarial examples**. Using this approach to provide examples for **adversarial training**.

Smoothness Prior with L_∞

- For problems with well-separated classes, we expect the classifier to assign the same class to x and $x' = x + \eta$ so long as $\|\eta\|_\infty \leq \epsilon$, where ϵ is small.
  - For $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$, $\|\mathbf{x}\|_\infty = \max_i |x_i|$.

The Linear Explanation of Adversarial Examples

- Let $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{x} + \eta$, the dot product between weight vector $\boldsymbol{w}$ and adversarial example x' is as follows

$$\hat{y}' = \boldsymbol{w}^T \boldsymbol{x}' = \boldsymbol{w}^T (\boldsymbol{x} + \eta) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{w}^T \eta \Rightarrow \hat{y}' - \hat{y} = \boldsymbol{w}^T \eta$$

  The adversarial perturbation causes the activation to grow by $\boldsymbol{w}^T \eta$.

The Linear Explanation of Adversarial Examples

- Let $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{x} + \eta$, the dot product between weight vector $\boldsymbol{w}$ and adversarial example x' is as follows

$$\hat{y}' = \boldsymbol{w}^T \boldsymbol{x}' = \boldsymbol{w}^T (\boldsymbol{x} + \eta) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{w}^T \eta \Rightarrow \hat{y}' - \hat{y} = \boldsymbol{w}^T \eta$$

  The adversarial perturbation causes the activation to grow by $\boldsymbol{w}^T \eta$.
- To generate adversarial example for x, we should maximize $\boldsymbol{w}^T \eta$, such that $\|\eta\|_\infty \leq \epsilon$. Therefore, we have the following maximization problem.

$$\begin{aligned} &\underset{\eta}{\text{argmax}} < \boldsymbol{w}, \eta > \\ &s.t. \quad \|\eta\|_\infty \leq \epsilon \end{aligned}$$

The Linear Explanation of Adversarial Examples

- Let $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{x} + \eta$, the dot product between weight vector $\boldsymbol{w}$ and adversarial example x' is as follows

$$\hat{y}' = \boldsymbol{w}^T \boldsymbol{x}' = \boldsymbol{w}^T (\boldsymbol{x} + \eta) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{w}^T \eta \Rightarrow \hat{y}' - \hat{y} = \boldsymbol{w}^T \eta$$

  The adversarial perturbation causes the activation to grow by $\boldsymbol{w}^T \eta$.

- To generate adversarial example for x, we should maximize $\boldsymbol{w}^T \eta$, such that $\|\eta\|_\infty \leq \epsilon$. Therefore, we have the following maximization problem.

$$\begin{aligned} &\underset{\eta}{\text{argmax}} < \boldsymbol{w}, \eta > \\ &s.t. \quad \|\eta\|_\infty \leq \epsilon \end{aligned}$$

  The solution to the above problem is $\eta^* = \epsilon.\text{sign}(\boldsymbol{w})$, we have

$$\hat{y}' - \hat{y} = \boldsymbol{w}^T \eta^* = \boldsymbol{w}^T \epsilon.\text{sign}(\boldsymbol{w}) = \epsilon \|\boldsymbol{w}\|_1$$

The Linear Explanation of Adversarial Examples

- Let $\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{x} + \eta$, the dot product between weight vector $\boldsymbol{w}$ and adversarial example x' is as follows

$$\hat{y}' = \boldsymbol{w}^T \boldsymbol{x}' = \boldsymbol{w}^T (\boldsymbol{x} + \eta) = \boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{w}^T \eta \Rightarrow \hat{y}' - \hat{y} = \boldsymbol{w}^T \eta$$

  The adversarial perturbation causes the activation to grow by $\boldsymbol{w}^T \eta$.
- To generate adversarial example for x, we should maximize $\boldsymbol{w}^T \eta$, such that $\|\eta\|_\infty \leq \epsilon$. Therefore, we have the following maximization problem.

$$\begin{aligned} &\underset{\eta}{\text{argmax}} < \boldsymbol{w}, \eta > \\ &s.t. \quad \|\eta\|_\infty \leq \epsilon \end{aligned}$$

  The solution to the above problem is $\eta^* = \epsilon.\text{sign}(\boldsymbol{w})$, we have

$$\hat{y}' - \hat{y} = \boldsymbol{w}^T \eta^* = \boldsymbol{w}^T \epsilon.\text{sign}(\boldsymbol{w}) = \epsilon \|\boldsymbol{w}\|_1$$

- If $\boldsymbol{w}$ has n dimensions and the average magnitude of an element of the weight vector is m, then the **activation will grow by** ϵmn**.** Thereby, as the dimension of the input increases, the value of $\hat{y}' - \hat{y}$ will grow.
- This explanation shows that a simple **linear model can have adversarial examples** if its input has **sufficient dimensionality**.

Linear Perturbation for Non-linear Models

The **linear view** of adversarial examples suggests a **fast** way of generating them.

- It is hypothesized that deep nets are **too linear** to resist adversarial perturbations (ReLU activation function).
- More nonlinear models such as **sigmoid or tanh** networks are carefully tuned to spend most of their time in the **non-saturating, more linear regime**.

Hence, we suppose Deep nets have **linear behavior** in the **vicinity** of each data point.

Linear Perturbation for Non-linear Models

The **linear view** of adversarial examples suggests a **fast** way of generating them.

- It is hypothesized that deep nets are **too linear** to resist adversarial perturbations (ReLU activation function).
- More nonlinear models such as **sigmoid or tanh** networks are carefully tuned to spend most of their time in the **non-saturating, more linear regime**.

Hence, we suppose Deep nets have **linear behavior** in the **vicinity** of each data point.

Recall: Taylor Series (Expansion)

Suppose n is a positive integer and $f : \mathbb{R} \to \mathbb{R}$ is n times differentiable at a point x_0. Then

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + R_n(x, x_0)$$

$$= f(x_0) + f'(x_0)(x - x_0) + \frac{f"(x_0)}{2}(x - x_0)^2 + \ldots$$

where the remainder R_n satisfies

$$R_n(x, x_0) = o(|x - x_0|^n) \text{ as } x \to x_0.$$

Definition: A sequence of numbers X_n is said to be $o(r_n)$ if $\frac{X_n}{r_n} \to 0$ as $n \to \infty$.

Linear Perturbation for Non-linear Models

The **linear view** of adversarial examples suggests a **fast** way of generating them.

- It is hypothesized that deep nets are **too linear** to resist adversarial perturbations (ReLU activation function).
- More nonlinear models such as **sigmoid or tanh** networks are carefully tuned to spend most of their time in the **non-saturating, more linear regime**.

Hence, we suppose Deep nets have **linear behavior** in the **vicinity** of each data point.

Consequently, we can linearly approximate classifier $f : \mathbb{R}^d \rightarrow \mathbb{R}$ around data point x_0 by **Taylor expansion**. We have:

$$f(x) = f(x_0) + (x - x_0)^T \nabla_x f(x)$$

Linear Perturbation for Non-linear Models

The **linear view** of adversarial examples suggests a **fast** way of generating them.

- It is hypothesized that deep nets are **too linear** to resist adversarial perturbations (ReLU activation function).
- More nonlinear models such as **sigmoid or tanh** networks are carefully tuned to spend most of their time in the **non-saturating, more linear regime**.

Hence, we suppose Deep nets have **linear behavior** in the **vicinity** of each data point.

Consequently, we can linearly approximate classifier $f : \mathbb{R}^d \rightarrow \mathbb{R}$ around data point x_0 by **Taylor expansion**. We have:

$$f(x) = f(x_0) + (x - x_0)^T \nabla_x f(x)$$

Let $x' = x_0 + \eta$, we get

$$f(x') = f(x + \eta) = f(x_0) + (\eta)^T \nabla_x f(x) \Rightarrow f(x') - f(x_0) = (\eta)^T \nabla_x f(x)$$

To maximize difference between $f(x)$ and $f(x')$, we should maximize $< \eta^T, \nabla_x f(x) >$. Given $\|\eta\|_\infty \leq \epsilon$, we have

$$\eta = \epsilon.sign(\nabla_x f(x))$$

Linear Perturbation for Non-linear Models

The **linear view** of adversarial examples suggests a **fast** way of generating them.

- It is hypothesized that deep nets are **too linear** to resist adversarial perturbations (ReLU activation function).
- More nonlinear models such as **sigmoid or tanh** networks are carefully tuned to spend most of their time in the **non-saturating, more linear regime**.

Hence, we suppose Deep nets have **linear behavior** in the **vicinity** of each data point.

Consequently, we can linearly approximate classifier $f : \mathbb{R}^d \rightarrow \mathbb{R}$ around data point x_0 by **Taylor expansion**. We have:

$$f(x) = f(x_0) + (x - x_0)^T \nabla_x f(x)$$

Let $x' = x_0 + \eta$, we get

$$f(x') = f(x + \eta) = f(x_0) + (\eta)^T \nabla_x f(x) \Rightarrow f(x') - f(x_0) = (\eta)^T \nabla_x f(x)$$

To maximize difference between $f(x)$ and $f(x')$, we should maximize $< \eta^T, \nabla_x f(x) >$. Given $\|\eta\|_\infty \leq \epsilon$, we have

$$\eta = \epsilon.sign(\nabla_x f(x))$$

We can replace classifier output with cost function J

$$\eta = \epsilon.sign(\nabla_x J(\theta, x, y))$$

Fast Gradient Sign Method (FGSM)

Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the label associated with $\boldsymbol{x}$ and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost used to train the neural network.

We can linearize the cost function around the current value of $\boldsymbol{\theta}$, obtaining an optimal max-norm constrained perturbation of

$$\boldsymbol{\eta} = \epsilon \, sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y))$$

We refer to this as the "**fast gradient sign method**" of generating adversarial examples.

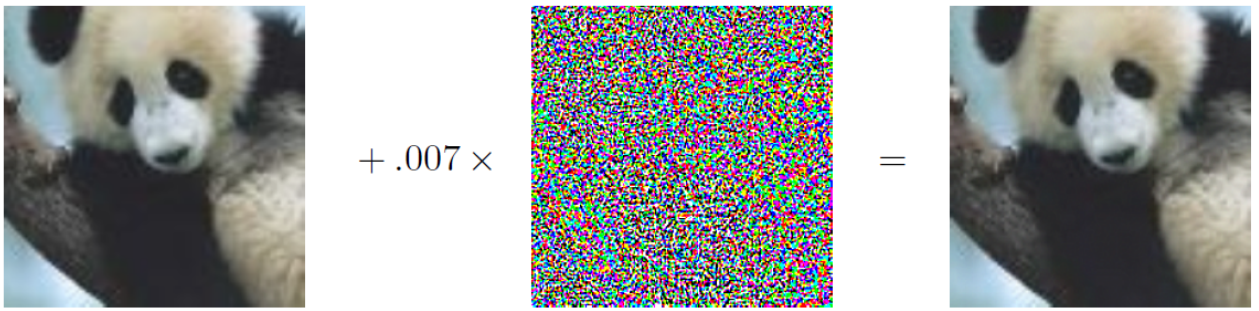


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

1. Add L_1 **regularization** to the cost function in order to reduce the size of $\|\boldsymbol{w}\|_1$.

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda \|\boldsymbol{w}\|_1$$

Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

1. Add L_1 **regularization** to the cost function in order to reduce the size of $\|\boldsymbol{w}\|_1$.

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda\|\boldsymbol{w}\|_1$$

2. Augment training set with adversarial examples (**Adversarial Training**).

Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

1. Add L_1 **regularization** to the cost function in order to reduce the size of $\|\boldsymbol{w}\|_1$.

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda\|\boldsymbol{w}\|_1$$

2. Augment training set with adversarial examples (**Adversarial Training**).

   If we train a single logistic regression model to recognize labels $y \in \{-1, 1\}$ with $P(y = 1) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ where $\sigma(z)$ is the logistic sigmoid function, then training consists of gradient descent on

$$\mathbb{E}_{x,y\sim P_{data}}\zeta(-y(\boldsymbol{w}^T\boldsymbol{x} + b))$$

   where $\zeta(z) = \log(1 + \exp(z))$ is the softplus function.

Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

1. Add L_1 **regularization** to the cost function in order to reduce the size of $\|\boldsymbol{w}\|_1$.

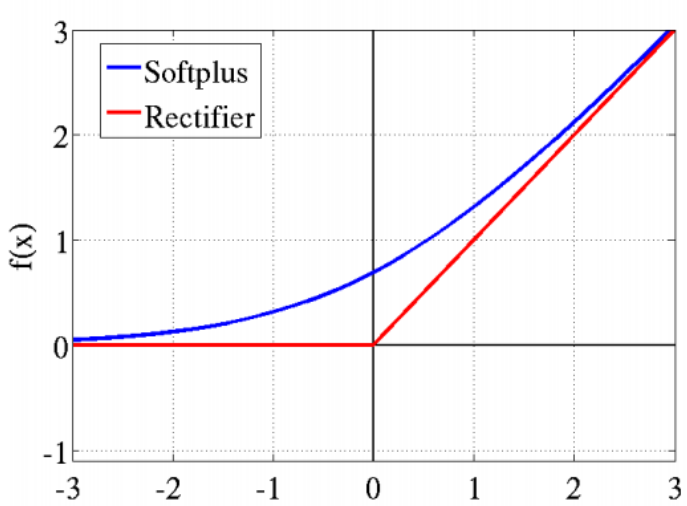
$$\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda\|\boldsymbol{w}\|_1$$

2. Augment training set with adversarial examples (**Adversarial Training**).

   If we train a single logistic regression model to recognize labels $y \in \{-1, 1\}$ with $P(y = 1) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ where $\sigma(z)$ is the logistic sigmoid function, then training consists of gradient descent on

$$\mathbb{E}_{x,y\sim P_{data}}\zeta(-y(\boldsymbol{w}^T\boldsymbol{x} + b))$$

   where $\zeta(z) = \log(1 + \exp(z))$ is the softplus function.



Defense Against Adversarial Examples

There are two solution to defend against adversarial examples

1. Add L_1 **regularization** to the cost function in order to reduce the size of $\|\boldsymbol{w}\|_1$.

$$\min_{\boldsymbol{w}} J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda\|\boldsymbol{w}\|_1$$

2. Augment training set with adversarial examples (**Adversarial Training**).

   If we train a single logistic regression model to recognize labels $y \in \{-1, 1\}$ with $P(y = 1) = \sigma(\boldsymbol{w}^T\boldsymbol{x} + b)$ where $\sigma(z)$ is the logistic sigmoid function, then training consists of gradient descent on

$$\mathbb{E}_{x,y\sim P_{data}}\zeta(-y(\boldsymbol{w}^T\boldsymbol{x} + b))$$

   where $\zeta(z) = \log(1 + \exp(z))$ is the softplus function.

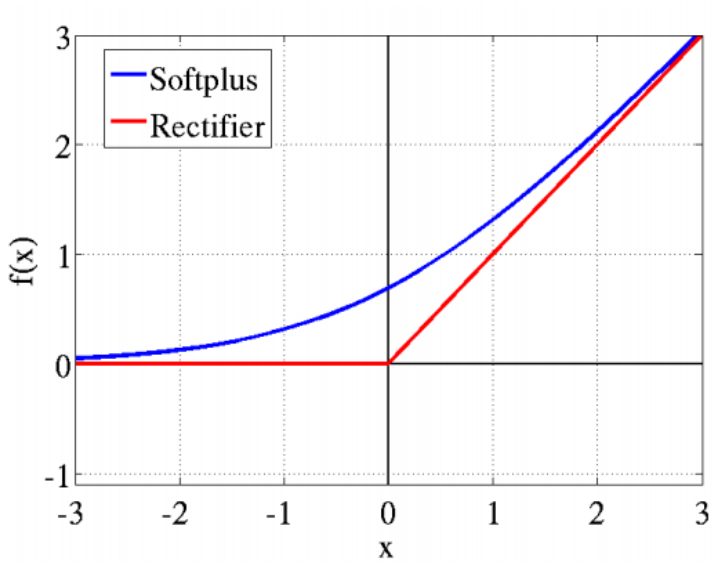
   We can derive a simple analytical form for training on FGSM generated adversarial examples. Note that $\boldsymbol{x}$ should move in direction of vector $-sign(\boldsymbol{w})$ to change the classification result. we have

$$\begin{aligned}
&\mathbb{E}_{x,y\sim P_{data}}\ \zeta(-y(\boldsymbol{w}^T(\boldsymbol{x} - \epsilon sign(\nabla_x \boldsymbol{w}^T\boldsymbol{x} + b)) + b)) \\
&= \mathbb{E}_{x,y\sim P_{data}}\ \zeta(-y(\boldsymbol{w}^T(\boldsymbol{x} - \epsilon sign(\boldsymbol{w})) + b)) \\
&= \mathbb{E}_{x,y\sim P_{data}}\ \zeta(-y(\boldsymbol{w}^T\boldsymbol{x} + b - \epsilon\|\boldsymbol{w}\|_1))
\end{aligned}$$

Defense Against Adversarial Examples

$$\mathbb{E}\,\zeta(-y(\boldsymbol{w}^T\boldsymbol{x} + b - \epsilon\|\boldsymbol{w}\|_1))$$

- This is somewhat **similar to L_1 regularization**.
- In adversarial training, the L_1 penalty is subtracted off the model's activation during training, rather than added to the training cost.
  - This means that the penalty can eventually **start to disappear if the model learns to make confident enough** predictions that ζ saturates. But, L_1 regularization is fixed during the course of training.
- L_1 regularization is **more pessimistic** to the adversarial training.



Adversarial Training

- Adversarial training with adversarial objective function based on the fast gradient sign method

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)), y)$$

where $\alpha = 0.5$.

Adversarial Training

- Adversarial training with adversarial objective function based on the fast gradient sign method

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)), y)$$

  where $\alpha = 0.5$.

- It reduces the error rate from 0.94% without adversarial training to 0.84% with adversarial training on MNIST.

Adversarial Training

- Adversarial training with adversarial objective function based on the fast gradient sign method

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)), y)$$

  where $\alpha = 0.5$.
- It reduces the error rate from 0.94% without adversarial training to 0.84% with adversarial training on MNIST.
- The model also became somewhat **resistant to adversarial examples**.
  - Without adversarial training, the model had an error rate of 89.4% on adversarial examples. With adversarial training, the error rate fell to 17.9%.

Adversarial Training

- Adversarial training with adversarial objective function based on the fast gradient sign method

$$\tilde{J}(\boldsymbol{\theta}, \boldsymbol{x}, y) = \alpha J(\boldsymbol{\theta}, \boldsymbol{x}, y) + (1 - \alpha) J(\boldsymbol{\theta}, \boldsymbol{x} + \epsilon sign(\nabla_x J(\boldsymbol{\theta}, \boldsymbol{x}, y)), y)$$

  where $\alpha = 0.5$.

- It reduces the error rate from 0.94% without adversarial training to 0.84% with adversarial training on MNIST.
- The model also became somewhat **resistant to adversarial examples**.
  - Without adversarial training, the model had an error rate of 89.4% on adversarial examples. With adversarial training, the error rate fell to 17.9%.
- The adversarial training procedure can be seen as **minimizing the worst case error** when the data is perturbed by an adversary.

Adversarial Training

They also found that the weights of the learned model changed significantly, with the weights of the adversarially trained model being significantly **more localized and interpretable**.

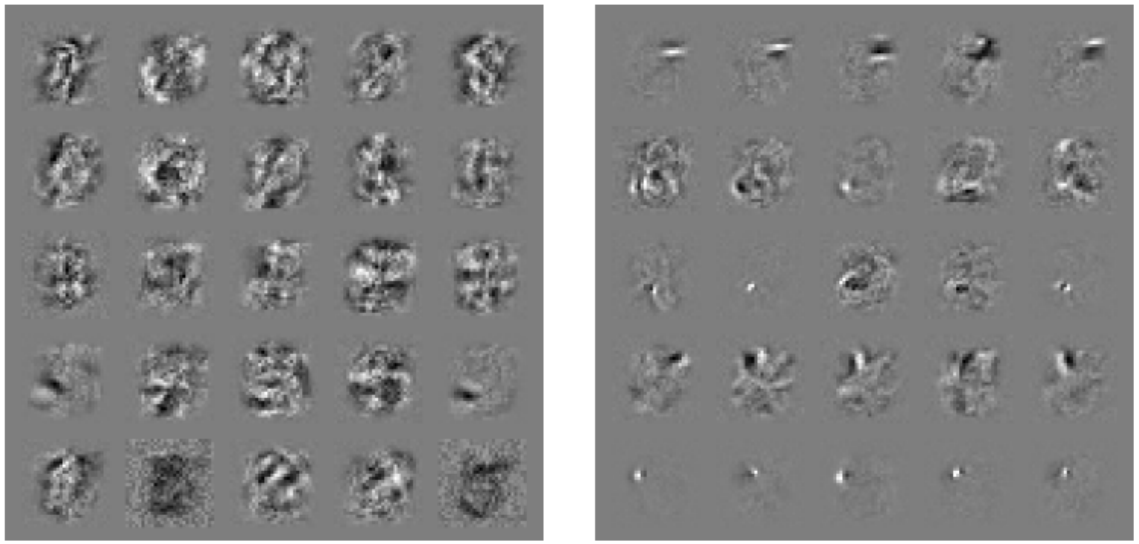
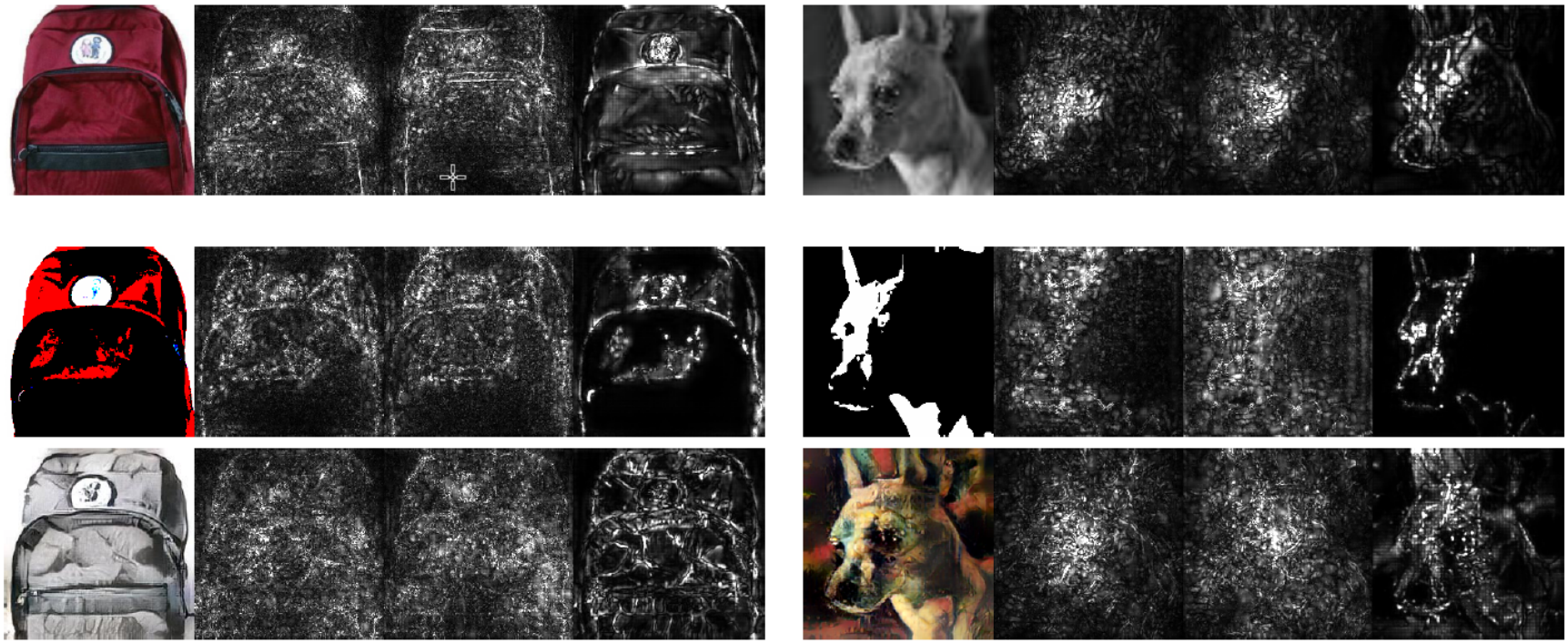


Figure 3: Weight visualizations of maxout networks trained on MNIST. Each row shows the filters for a single maxout unit. Left) Naively trained model. Right) Model with adversarial training.

Adversarial Training

We find that **adversarial training alleviates the texture bias** of standard CNNs when trained on object recognition tasks, and helps CNNs **learn a more shape-biased representation**. This finding partially explains why adversarially trained-CNNs tends to be more robust than standard CNNs.



(a) Images from Caltech-256 (b) Images from Tiny ImageNet

Figure 2. Sensitivity maps based on SmoothGrad (Smilkov et al., 2017) of three models on images under saturation, and stylizing. From top to bottom, Original, Saturation 1024 and Stylizing. For each group of images, from left to right, original image, sensitivity maps of standard CNN, underfitting CNN and PGD-l_∞ AT-CNN.

Interpreting Adversarially Trained Convolutional Neural Networks, Zhang, ICML 2019.

Why Do Adversarial Examples Generalize?

By tracing out different values of ϵ we see that adversarial examples occur in **contiguous regions** of the 1-D subspace defined by the fast gradient sign method, **not in fine pockets**.

- This explains why adversarial examples are abundant and why an example misclassified by one classifier has a fairly high prior probability of being misclassified by another classifier.

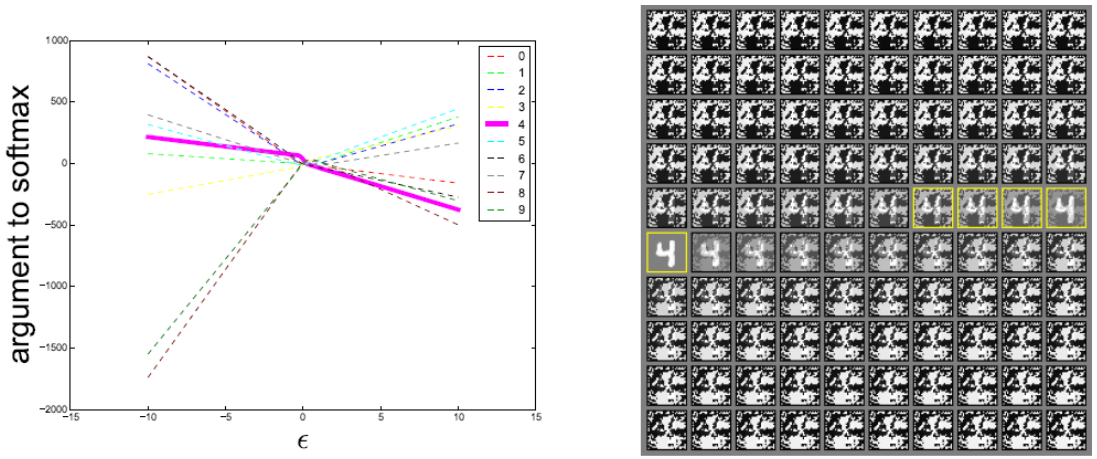
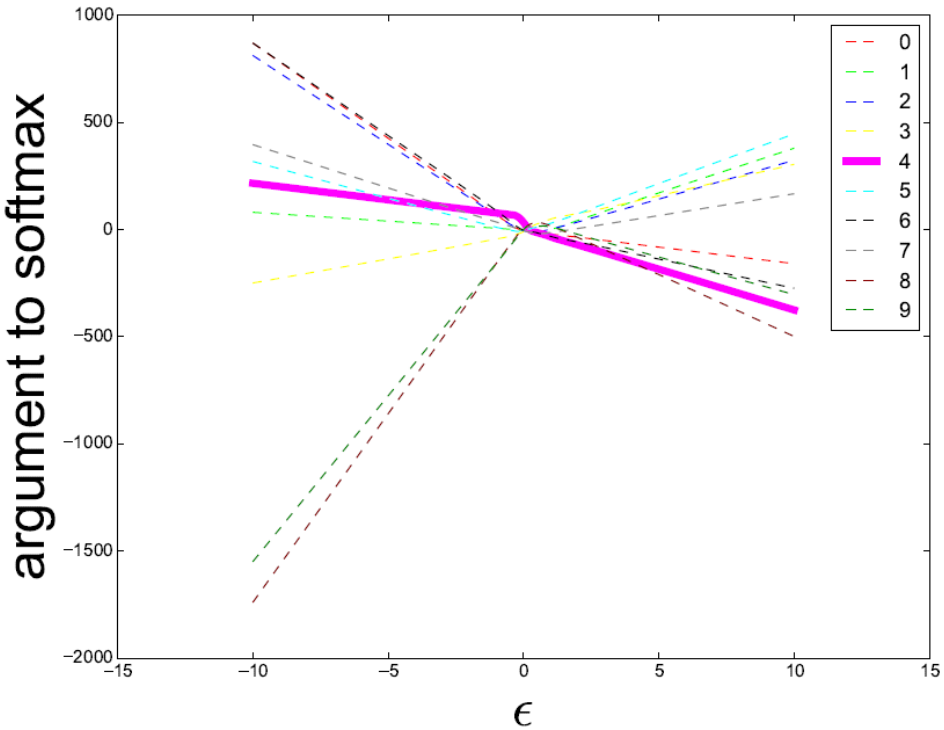


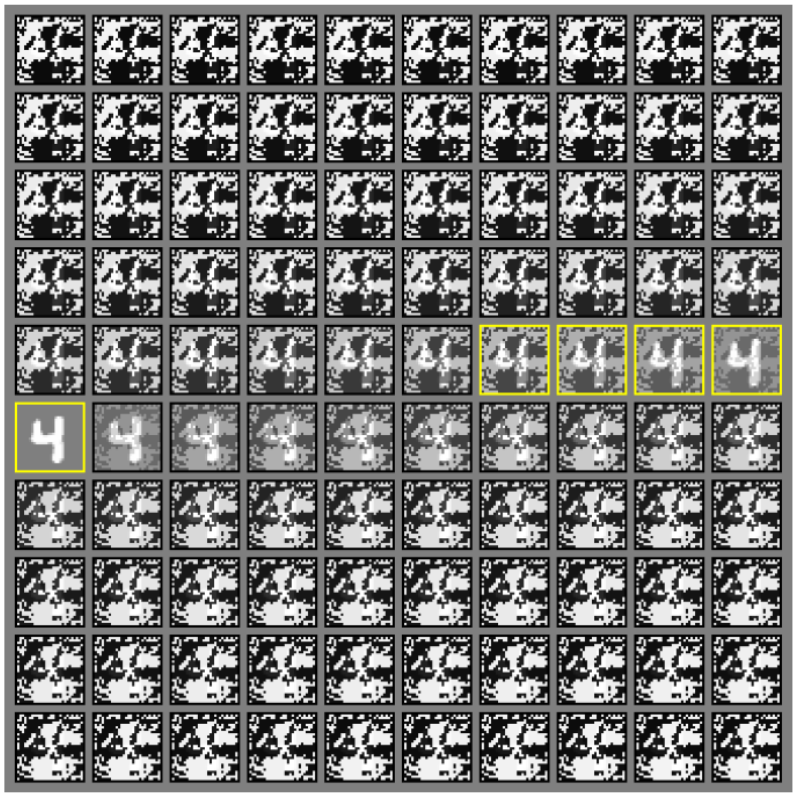
Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where $\boldsymbol{x}$ occurs in the data. Most of $\mathbb{R}^n$ consists of adversarial examples and *rubbish class examples* (see the appendix). This plot was made from a naively trained maxout network. Left) A plot showing the argument to the softmax layer for each of the 10 MNIST classes as we vary ϵ on a single input example. The correct class is 4. We see that the unnormalized log probabilities for each class are conspicuously piecewise linear with ϵ and that the wrong classifications are stable across a wide region of ϵ values. Moreover, the predictions become very extreme as we increase ϵ enough to move into the regime of rubbish inputs. Right) The inputs used to generate the curve (upper left = negative ϵ, lower right = positive ϵ, yellow boxes indicate

Why Do Adversarial Examples Generalize?

By tracing out different values of ϵ we see that adversarial examples occur in **contiguous regions** of the 1-D subspace defined by the fast gradient sign method, **not in fine pockets**.

- This explains why adversarial examples are abundant and why an example misclassified by one classifier has a fairly high prior probability of being misclassified by another classifier.





Observations

- Adversarial examples can be explained as a property of high-dimensional dot products. They are a result of models being **too linear, rather than too nonlinear**.
- The **direction of perturbation**, rather than the specific point in space, matters most.
- Because it is the direction that matters most, adversarial perturbations **generalize** across different clean examples.

Potemkin village - Clever Hans

- These results suggest that classifiers based on modern machine learning techniques, even those that obtain excellent performance on the test set, are not learning the true underlying concepts that determine the correct output label.
- Instead, these algorithms have built a **Potemkin village that works well on naturally occuring data, but is exposed as a fake when one visits points in space that do not have high probability in the data distribution.**
- Clever Hans was a horse that was claimed to have performed arithmetic and other intellectual tasks. After a formal investigation in 1907, psychologist Oskar Pfungst demonstrated that the horse was not actually performing these mental tasks, but was watching the reactions of his trainer.

References

- C. Szegedy, W. Zaremba, I. Sutskever, et al., ”Intriguing properties of neural networks,” in 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014.
- I. Goodfellow, J. Shlens, C. Szegedy, ”Explaining and Harnessing Adversarial Examples” in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015.
- D. Baehrens, T. Schroeter, S. Harmeling, et al., ”How to explain individual classification decisions.” The Journal of Machine Learning Research 11 (2010): 1803-1831.