Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

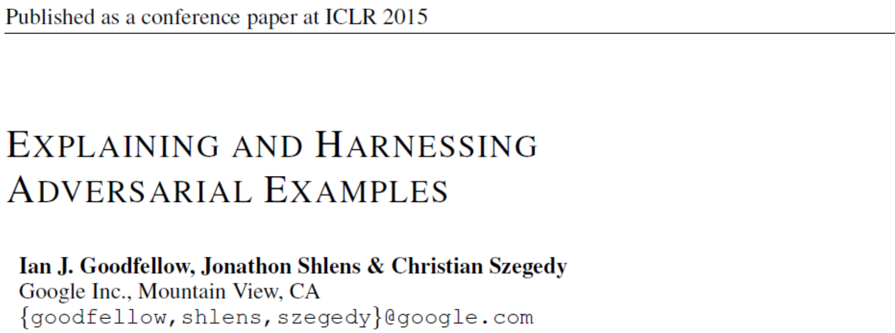
Sharif University of Technology
Computer Engineering Department (CE)
Data and Network Security Lab (DNSL)

October 15, 2024

Today’s Agenda

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 $\mathbf{x} = [x_1, x_2, \ldots, x_d]^T$, $\|\mathbf{x}\|_\infty = \underset{i}{max}\ |x_i|$.

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.

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

  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.

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

  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 $f : \mathbb{R}^d \to \mathbb{R}$ 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 $f : \mathbb{R}^d \to \mathbb{R}$ 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.

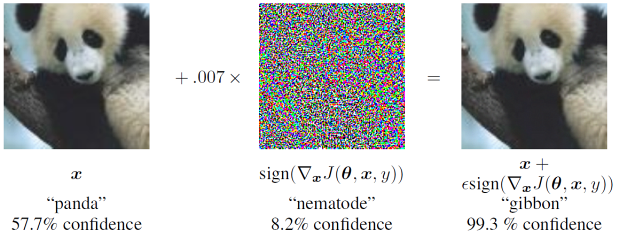

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

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 $||\boldsymbol{w}||_1$.

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

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 $\|\boldsymbol{w}\|_1$.

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

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 $||\boldsymbol{w}||_1$.

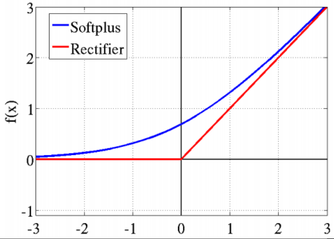
$$\underset{\boldsymbol{w}}{min}\ J(\boldsymbol{w}, \boldsymbol{x}, y) = \zeta(f(\boldsymbol{x}), y) + \lambda||\boldsymbol{w}||_1$$

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.

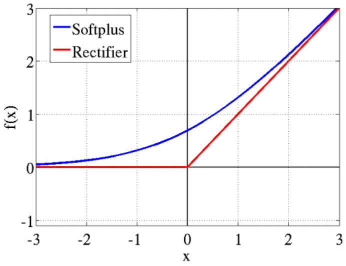
   We can derive a simple analytical form for training on FGSM generated adversarial examples. Note that $\boldsymbol{x}$ should move in direction of vector $-sign(\boldsymbol{w})$ to change the classification result. we have

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

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**.

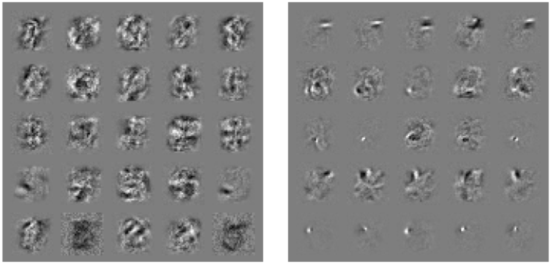
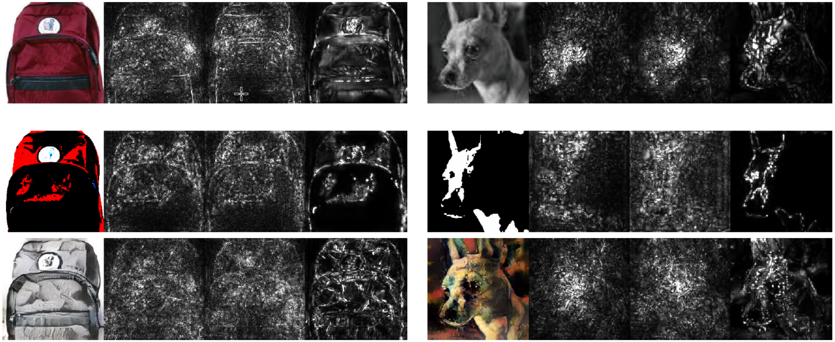


Figure 3: Weight visualizations of maxout networks trained on MNIST. Each row shows the filters for a single maxout unit. Left) Naively trained model. Right) Model with adversarial training.

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.



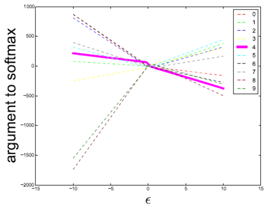
(a) Images from Caltech-256 (b) Images from Tiny ImageNet

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.

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.



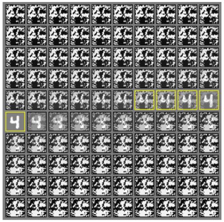
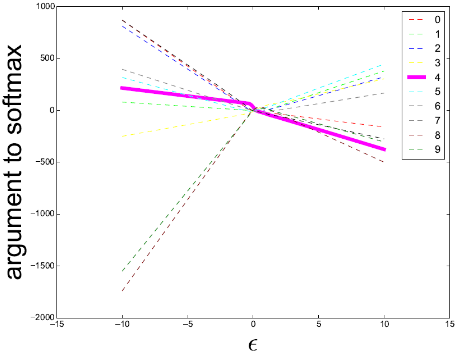


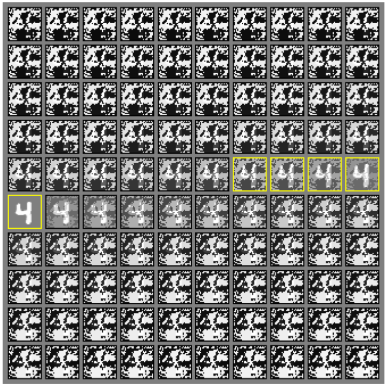
Figure 4: By tracing out different values of ϵ, we can see that adversarial examples occur reliably for almost any sufficiently large value of ϵ provided that we move in the correct direction. Correct classifications occur only on a thin manifold where $\boldsymbol{x}$ occurs in the data. Most of $\mathbb{R}^n$ 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.