Adversarial Example

Model $f(x) \rightarrow y$
data point $\rightarrow$ category

Model $f$

bagle

($\sim 80\%$ accuracy)
Adversarial Example

Training

Model $f(x) \rightarrow y$

Data point $\rightarrow$ Category

Attack: small perturbation

Model $f$

Bagle ($\sim 80\%$ accuracy)

Adversarial example

Model $f$

Grand piano
($\sim 0\%$ accuracy)
More Examples

- **Robustness** is critical in real systems

- **Real world adversarial examples:**
  - Attack stop sign in real world (Evtimov et al., 2017)
  - Adversarial turtle (Athalye et al., 2017)

![Stop sign](image1.png) + 0.001x = ![Teddy bear](image2.png)
Robustness is critical in real systems.

Real world adversarial examples:
- Attack stop sign in real world (Evtimov et al., 2017)
- Adversarial turtle (Athalye et al., 2017)
Research Questions

Questions:

**Attack**: How to generate adversarial example?

**Verification**: evaluate the robustness of deep networks

**Defense**: how to improve robustness?
Notations and Attack Procedure

original example \( x_0 \) \( \xrightarrow{\Delta} \) small distortion \( x^* \) \( \xrightarrow{f} \) correct label \( y_0 = f(x_0) \)

adversarial example \( x^* \) \( \xrightarrow{f} \) \( y^* = f(x^*) \)
Attack as an optimization problem

Input image: \( x_0 \in \mathbb{R}^d \)
Adversarial image: \( x^* \in \mathbb{R}^d \)

\[
x^* = \arg \min_x \text{Dis}(x, x_0) + c \cdot g(x)
\]
Attack as an optimization problem

Input image: $\mathbf{x}_0 \in \mathbb{R}^d$
Adversarial image: $\mathbf{x} \in \mathbb{R}^d$

$$\arg\min_{\mathbf{x}} \text{Dis}(\mathbf{x}, \mathbf{x}_0) + c \cdot g(\mathbf{x})$$

- $\text{Dis}(\mathbf{x}, \mathbf{x}_0)$: the distortion, e.g., $\|\mathbf{x} - \mathbf{x}_0\|_2^2$
Attack as an optimization problem

Input image: \( x_0 \in \mathbb{R}^d \)
Adversarial image: \( x \in \mathbb{R}^d \)

\[
\arg \min_x \| x - x_0 \|_2^2 + c \cdot g(x)
\]

- \( \text{Dis}(x, x_0) \): the distortion
- \( g(x) \): loss to measure the **successfulness of attack**
Attack as an optimization problem

Input image: \( \mathbf{x}_0 \in \mathbb{R}^d \)
Adversarial image: \( \mathbf{x}^* \in \mathbb{R}^d \)

\[
\arg \min_{\mathbf{x}} \| \mathbf{x} - \mathbf{x}_0 \|^2_2 + c \cdot g(\mathbf{x})
\]

- \( \text{Dis}(\mathbf{x}, \mathbf{x}_0) \): the distortion
- \( g(\mathbf{x}) \): loss to measure the \text{successfulness of attack}
- \( c \geq 0 \) controls the trade-off
- See (Carlini & Wagner, 2017)
Define successfulness of attack
Define successfulness of attack

Untargeted attack: success if $f_j(x) \neq y_0$

$$g(x) = \max\{f_{y_0}(x) - \max_{j \neq y_0} f_j(x), 0\}$$
Define successfulness of attack

- Targeted attack: success if $f_j(x') = t$

$$g(x) = \max \{ \max_{j \neq t} f_j(x) - f_t(x), 0 \}$$
White box setting

\[ x^* = \arg \min_x \ Dis(x, x_0) + c \cdot g(x) \] (1)

- Model \( f(\cdot) \) is revealed to attacker
  - \( \Rightarrow \) gradient of \( g(x) \) can be computed by back-propagation
  - \( \Rightarrow \) attacker minimizes (1) by gradient descent
White box setting

\[ x^* = \arg \min_x \text{Dis}(x, x_0) + c \cdot g(x) \]  \hspace{1cm} (1)

- Model \( f(\cdot) \) is revealed to attacker
  \( \Rightarrow \) gradient of \( g(x) \) can be computed by back-propagation
  \( \Rightarrow \) attacker minimizes (1) by gradient descent
- Can be generalized to other definition of \( \text{Dis}(x, x_0) \)
  (e.g., \( \ell_1 \) penalty, Elastic net, \( \cdots \))

Attack Image Captioning

- Usually use CNN+RNN network

(Figure from Show and Tell, Vinyals et al, 2014)
**Attack Image Captioning**

- Usually use **CNN+RNN network**

(Figure from Show and Tell, Vinyals et al, 2014)

- Define the **loss on sequence output** for different attacks
Black Box Setting

- In practice, the deep network structure/parameters are not revealed to attackers
  
  \[ \text{cannot compute gradient } \nabla f(x) \]
In practice, the deep network structure/parameters are not revealed to attackers.

**cannot compute gradient** $\nabla f(x)$

Attacker can **query** the ML model and get the **probability output**

---

**Black box (can’t see f)**
In practice, the deep network structure/parameters are not revealed to attackers.

Attacker cannot compute gradient $\nabla f(x)$.

Attacker can query the ML model and get the probability output.

Previous approach (Papernot et al., 2016):

Train a substitute model using $(x_1, f(x_1)), \cdots, (x_q, f(x_q))$

Attack this substitute model.
Black Box Setting

\[ \arg \min_x \text{Dis}(x, x_0) + c \cdot g(x) \]

- We can actually solve the same optimization problem!

ZOO: Zeroth Order Optimization based Black-box Attacks. [C, Z, S, Y, H], CCS AI-Security 2017
Black Box Setting

\[
\arg \min_x \ \text{Dis}(x, x_0) + c \cdot g(x)
\]

- We can actually solve the same optimization problem!
- Symmetric difference quotient to estimate gradient:

\[
\hat{g}_i \approx \frac{\partial F(x)}{\partial x_i} \approx \frac{F(x + \epsilon e_i) - F(x - \epsilon e_i)}{2\epsilon}
\]

ZOO: Zeroth Order Optimization based Black-box Attacks. [C, Z, S, Y, H], CCS AI-Security 2017
Black Box Setting

\[
\arg \min_x \text{Dis}(x, x_0) + c \cdot g(x)
\]

- We can actually solve the same optimization problem!
- Symmetric difference quotient to estimate gradient:

\[
\hat{g}_i \approx \nabla_i g(x) \approx \frac{g(x + \epsilon e_i) - g(x - \epsilon e_i)}{2 \epsilon}
\]

- Zeroth order gradient descent

\[
\begin{bmatrix}
\hat{g}_1 \\
\vdots \\
\hat{g}_d
\end{bmatrix}
\]

\[
x \leftarrow x - \eta
\]

ZOO: Zeroth Order Optimization based Black-box Attacks. [C, Z, S, Y, H], CCS AI-Security 2017
However, need $O(d)$ queries to estimate gradient.
Black Box Setting

- However, Need $O(d)$ queries to estimate gradient
  - ImageNet: $d = 299 \times 299 \times 3 > 268K$
  - 100 iterations $\Rightarrow$ 26 million queries
Black Box Setting

- However, Need $O(d)$ queries to estimate gradient
  - ImageNet: $d = 299 \times 299 \times 3 > 268K$
  - 100 iterations $\Rightarrow$ 26 million queries
- Reduce number of queries:
  - Stochastic Coordinate descent: update a small set of coordinates at each time
  - Greedy approach: select important coordinates first
  - Attack-space dimension reduction + hierarchical attack

![Graphs showing pixel coordinates for different image sizes: 32x32, 64x64, 128x128]
100% success rate, similar distortion with white box attack.

<table>
<thead>
<tr>
<th></th>
<th>MNIST</th>
<th></th>
<th>CIFAR10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Success Rate</td>
<td>Avg. $L_2$</td>
<td>Avg. Time (per attack)</td>
</tr>
<tr>
<td>White-box</td>
<td>100 %</td>
<td>2.00661</td>
<td>0.53 min</td>
</tr>
<tr>
<td>Black-box (Substitute Model)</td>
<td>26.74 %</td>
<td>5.272</td>
<td>0.80 min (+ 6.16 min)</td>
</tr>
<tr>
<td>Proposed black-box (ZOO)</td>
<td><strong>98.9 %</strong></td>
<td>1.987068</td>
<td>1.62 min</td>
</tr>
</tbody>
</table>
Black-box Attack on ImageNet (Inception-v3)

- grand piano + black-box attack = Dutch oven
- daisy + black-box attack = cup
- zebra + black-box attack = hartebeest
- ptarmigan + black-box attack = black grouse
- cinema + black-box attack = church
- washbasin + black-box attack = soap dispenser
Questions:

Attack: How to attack?

**Verification:** Evaluate the robustness of your model

Defense: how to improve robustness?
Measuring Robustness

- How to measure the “robustness” of a neural network?
Measuring Robustness

- How to measure the “robustness” of a neural network?
- The measurement should be independent to attack algorithms!
How to measure the “robustness” of a neural network?

The measurement should be independent to attack algorithms!

Robustness: For an example $x_0$, find $r$ such that

$$f(x_0 + \Delta) \text{ is correct for all } \|\Delta\| < r$$
**Theorem:** For a model $f(\cdot)$ and an example $x_0$,

$$f(x_0 + \Delta) = y \text{ for all } \|\Delta\| < \min_{j\neq c} \frac{f_y(x_0) - f_j(x_0)}{L_j},$$

where $L_j = \max\{\|\nabla f_y(x) - \nabla f_j(x)\| : x \in S\}$

How to evaluate $L_j$?

$$L_j = \max \{ \| \nabla h(x) \| : x \in S \}$$

- First attempt: Solve as an optimization problem

Doesn't work...: $\nabla f(\cdot)$ is non-convex and often non-smooth
How to evaluate $L_j$?

\[ L_j = \max\{\|\nabla h(x)\| : x \in S\} \]

- First attempt: Solve as an optimization problem
- Doesn’t work: $\nabla f(\cdot)$ is non-convex and often non-smooth
Sampling Approach

\[ L_j = \max \{ \| \nabla h(x) \| : x \in S \} \]

- Naive Sampling Algorithm
  - Sample \( x_1, x_2, \cdots, x_T \in S \) and select max
 Sampling Approach

\[ L_j = \max\{\|\nabla h(x)\| : x \in S\} \]

- Naive Sampling Algorithm
  - Sample \(x_1, x_2, \ldots, x_T \in S\) and select max

- Dimension is too high
  \[ \Rightarrow \text{Non-meaningful solution even with } > 1 \text{ million samples} \]
Extreme Value Theory

Extreme Value Theorem (Fisher-Tippett-Gnedenko Theorem)

Let $X_1, \cdots, X_n$ be iid random variables and $M_n = \max\{X_1, \cdots, X_n\}$. Define the cdf of $M_n$:

$$P(M_n \leq x) = G_n(x).$$

When $n \to \infty$, $G_n$ belongs to either the Gumbel, Frechet, or Reverse Weibull family.
Evaluate robustness

Computing CLEVER score

\( P \leftarrow \{ \varphi \} \)
- For \( i = 1, 2, \cdots, M \)
  - Randomly sample \( x_1, \cdots, x_N \in S \)
  - Compute \( b \leftarrow \max_{i=1, \cdots, N} \| \nabla h(x_i) \| \)
- \( P \leftarrow P \cup b \)

\( \hat{a} \leftarrow \text{MLE of location parameter of reverse Weibull distribution on } S \)

\[
\begin{align*}
r &\geq \frac{h(x_0)}{L_j} \approx \frac{h(x_0)}{\hat{a}} \quad \text{(CLEVER score)}
\end{align*}
\]
Experiments

- CW Attack: upper bound of \( r \)
- CLEVER: estimated lower-bound of \( r \)
- SLOPE: previous algorithm for estimating \( r \)

<table>
<thead>
<tr>
<th>Dataset/Model</th>
<th>CW Attack</th>
<th>CLEVER</th>
<th>SLOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST-CNN</td>
<td>1.504</td>
<td>0.987</td>
<td>7.921</td>
</tr>
<tr>
<td>MNIST-DD</td>
<td>1.542</td>
<td>1.330</td>
<td>9.646</td>
</tr>
<tr>
<td>MNIST-BReLU</td>
<td>1.404</td>
<td>1.583</td>
<td>23.548</td>
</tr>
<tr>
<td>CIFAR-CNN</td>
<td>0.188</td>
<td>0.123</td>
<td>2.195</td>
</tr>
<tr>
<td>CIFAR-DD</td>
<td>0.296</td>
<td>0.220</td>
<td>3.083</td>
</tr>
<tr>
<td>CIFAR-BReLU</td>
<td>0.152</td>
<td>0.052</td>
<td>1.564</td>
</tr>
<tr>
<td>Resnet-50</td>
<td>0.212</td>
<td>0.134</td>
<td>-</td>
</tr>
<tr>
<td>MobileNet</td>
<td>0.190</td>
<td>0.144</td>
<td>-</td>
</tr>
</tbody>
</table>

DD, BReLU: defensive algorithms
Estimated lower bound can be sometimes larger than $r$

A safe zone only in probability

Estimate of $\beta$
Can we do better for ReLU Network?

- State-of-the-art networks use ReLU activation:
  \[ \theta(x) = \max(0, x) \]

- Goal: provide a certified, meaningful and computational efficient lower bound of \( r \)
Previous Work

- ReLUplex (Katz et al., 2017): combinatorial optimization,
  Network with \textbf{300} neurons need \textbf{3 hours}
- Linear Programming (Kolter & Wong, 2017):
  Network with \textbf{5120} neurons need \textbf{8 hours}
- State-of-the-art networks have \textbf{millions} of neurons
A linear time algorithm

Assume perturbation is bounded:

\[ x_i = x_i + \delta_i, \quad -0.1 \leq \delta \leq -0.1 \]

\[ f_1(x) = W_4 \theta(W_3 \theta(W_2 \theta(W_1(x + \delta)))) \]
A linear time algorithm

- Activated neuron: input $\geq 0$ (for all perturbations)
  
  \[ \text{output} = \max(0, \text{input}) = \text{input} \]

- All neurons activated $\Rightarrow$ Linear neural network
  
  \[ f_1(x + \delta) = W_4 W_3 W_2 W_1 (x + \delta) \]
A linear time algorithm

- **Inactivated** neuron: input $< 0$ for all perturbations

  \[
  \text{output} = \max(0, \text{input}) = 0
  \]

- Remove those nodes, still linear
A linear time algorithm

- **Unsure neuron**: input can be positive or negative
- **Deal with both cases**
  - $\Rightarrow$ exponential time algorithms (previous approaches)
A linear time algorithm

Assume we know $\ell \leq \text{input} \leq u$, how to approximate?
A linear time algorithm

Assume we know $\ell \leq \text{input} \leq u$, how to approximate?

\[
\frac{u}{u-\ell} \times \text{input} \leq \max(0, \text{input}) \leq \frac{u}{u-\ell} \times (\text{input} - \ell)
\]
A linear time algorithm

Replace by a linear function with an added bias term
A linear time algorithm

Form the **linear approximation** layer by layer

Now Upper/lower bounds of $f_j(x)$ can be easily computed
Moreover

- Same time complexity with forward-propagation!
Moreover

- Same time complexity with \textit{forward-propagation}!
- We have another algorithm to compute Lipchitz constant (gradient upperbound)
  
  Similar to \textit{back-propagation}!
Experiments

- Fast-Lin: Our method (python)
- ReluPlex: Exponential time algorithm for computing exact bound (Katz et al., 2017)
- LP: Linear programming for lower bound (Kolter and Wong, 2017)

<table>
<thead>
<tr>
<th>Network</th>
<th>Method</th>
<th>Time</th>
<th>$\delta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNIST 2*20</td>
<td>Fast-Lin</td>
<td>0.028</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td>ReluPlex</td>
<td>133.78</td>
<td>0.41</td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>0.213</td>
<td>0.20</td>
</tr>
<tr>
<td>MNIST 3*20</td>
<td>Fast-Lin</td>
<td>0.10</td>
<td>0.16</td>
</tr>
<tr>
<td></td>
<td>ReluPlex</td>
<td>17046</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>0.25</td>
<td>0.17</td>
</tr>
<tr>
<td>MNIST 4*1024</td>
<td>Fast-Lin</td>
<td>3.7</td>
<td>0.007</td>
</tr>
<tr>
<td></td>
<td>LP</td>
<td>1094</td>
<td>0.007</td>
</tr>
<tr>
<td>CIFAR 5*2048</td>
<td>Fast-Lin</td>
<td>30.4</td>
<td>0.000084</td>
</tr>
</tbody>
</table>
Questions:

Attack: How to attack?
Verification: Evaluate the robustness of your model
Defense: how to improve robustness?
Defense from Adversarial Attacks

- Adversarial retraining (Goodfellow et al., 2015):
  Adding adversarial examples into training set
- Robust optimization + BReLu (Zantedeschi et al., 2017)
  \[
  \min_w \mathbb{E}_{(x,y) \sim D} \mathbb{E}_{\Delta x \sim N(0,\sigma^2)} \text{loss}(f_w(x + \Delta x), y)
  \]
- Security community: detect adversarial examples
  MagNet: (Meng and Chen, 2017), (Li and Li, 2017), ...
Defense from Adversarial Attacks

- Adversarial retraining (Goodfellow et al., 2015):
  Adding adversarial examples into training set

- Robust optimization + BReLU (Zantedeschi et al., 2017)

\[
\min_w \mathbb{E}_{(x,y) \sim D} \mathbb{E}_{\Delta x \sim N(0,\sigma^2)} \text{loss}(f_w(x + \Delta x), y)
\]

- Security community: detect adversarial examples
  MagNet: (Meng and Chen, 2017), (Li and Li, 2017), · · ·

- However, they are still vulnerable if we choose a better way to attack (Carlini and Wagner, 2017, MagNet and “Efficient Defenses Against Adversarial Attacks” are Not Robust to Adversarial Examples)
Adding randomness to the model

- All the attacks are based on gradient computation
Adding randomness to the model

- All the attacks are based on gradient computation
- Idea I: Adding randomness to fool the attackers
  \[ f(x) \rightarrow f_\epsilon(x) \text{ where } \epsilon \text{ is random} \]
- Resistant to Black-box attack: gradient cannot be estimated
  \[ \frac{f_{\epsilon_1}(x + he_i) - f_{\epsilon_2}(x - he_i)}{2h} \not\approx \nabla_i f(x) \]
Adding randomness to the model

- Naive approach: adding randomness only in the testing phase
  prediction accuracy 87% $\rightarrow$ 20% (Cifar-10 with VGG)
Ensemble

- How to boost the accuracy?
Ensemble

- How to boost the accuracy?
- Idea II: Ensemble can improve the robustness of model:
  
  ensemble $T$ models $f_{\epsilon_1}(x), \cdots, f_{\epsilon_T}(x)$

- Our approach: Random + Ensemble
Our approach (RSE)

- Prediction: Ensemble of random models

\[ p = \sum_{j=1}^{T} f_{\epsilon_j}(w, x), \quad \text{and predict } \hat{y} = \arg\max_k p_k \]
Our approach (RSE)

- Prediction: Ensemble of random models
  \[ p = \sum_{j=1}^{T} f_{\epsilon_j}(w, x), \quad \text{and predict } \hat{y} = \arg \max_k p_k \]

- Training: minimizing an upper bound
  \[ E_{(x, y) \sim D} E_{\epsilon \sim N} \text{loss}(f_{\epsilon}(x), y) \geq E_{(x, y) \sim D} \text{loss}(E_{\epsilon \sim N} f_{\epsilon}(x), y) \]

- Training by SGD: sample \((x, y), \epsilon\) and conduct
  \[ w \leftarrow w - \eta \nabla_w \text{loss}(f_{\epsilon}(x), y) \]
Our approach (RSE)

- Prediction: Ensemble of random models
  \[ p = \sum_{j=1}^{T} f_{\epsilon_j}(w, x), \text{ and predict } \hat{y} = \arg \max_{k} p_k \]

- Training: minimizing an upper bound
  \[ E_{(x,y) \sim D} \epsilon \sim \mathcal{N} \text{loss}(f_{\epsilon}(x), y) \geq E_{(x,y) \sim D} \text{loss}(E_{\epsilon \sim \mathcal{N}} f_{\epsilon}(x), y) \]

- Training by SGD: sample \((x, y), \epsilon\) and conduct
  \[ w \leftarrow w - \eta \nabla_w \text{loss}(f_{\epsilon}(x), y) \]

- Implicit regularizing Lipchitz constant:
  \[ E_{\epsilon \sim \mathcal{N}(0, \sigma^2)} \text{loss}(f_{\epsilon}(w, x_i), y_i) \approx \text{loss}(f_0(w, x_i), y_i) + \frac{\sigma^2}{2} L_{\ell} f_0, \]
Experiments

- We fix $\sigma_{init} = 0.4$ and $\sigma_{inner} = 0.1$ for all the datasets/networks.
Targeted attack

Original image:

Targeted attack

<table>
<thead>
<tr>
<th>plain</th>
<th>dd</th>
<th>adv</th>
<th>brelu</th>
<th>rse</th>
</tr>
</thead>
<tbody>
<tr>
<td>bird</td>
<td>car</td>
<td>cat</td>
<td>deer</td>
<td>dog</td>
</tr>
<tr>
<td>frog</td>
<td>horse</td>
<td>plane</td>
<td>truck</td>
<td></td>
</tr>
</tbody>
</table>
References


