Black-box Adversarial Examples

A. M. Sadeghzadeh, Ph.D.

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

May 2, 2023

Today's Agenda

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

- In this setting, back propagation for **gradient computation of the targeted model is prohibited**.

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

- In this setting, back propagation for **gradient computation of the targeted model is prohibited**.
- Threat model
  - **Score-based** (the adversary has access to the target model scores)
  - **Decision-based** (the adversary has only access to the target model label)

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

- In this setting, back propagation for **gradient computation of the targeted model is prohibited**.
- Threat model
  - **Score-based** (the adversary has access to the target model scores)
  - **Decision-based** (the adversary has only access to the target model label)
- Types
  - **Transfer-based** and **Query-based**

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

- In this setting, back propagation for **gradient computation of the targeted model is prohibited**.
- Threat model
  - **Score-based** (the adversary has access to the target model scores)
  - **Decision-based** (the adversary has only access to the target model label)
- Types
  - **Transfer-based** and **Query-based**

Transfer-based

- Create a surrogate model with high **fidelity** to the taget model.
- Generate adversarial examples on the surrogate model using **white-box** attacks.
- Then, **transfer** pregenerated adversarial examples to the target model.

Black-box Adversarial Examples

The only capability of the black-box adversary is to observe the **output** given by the target model to **chosen inputs**.

- In this setting, back propagation for **gradient computation of the targeted model is prohibited**.
- Threat model
  - **Score-based** (the adversary has access to the target model scores)
  - **Decision-based** (the adversary has only access to the target model label)
- Types
  - **Transfer-based** and **Query-based**

Transfer-based

- Create a surrogate model with high **fidelity** to the taget model.
- Generate adversarial examples on the surrogate model using **white-box** attacks.
- Then, **transfer** pregenerated adversarial examples to the target model.

Query-based

- Based on the target model responses for consecutive queries
  - Gradient estimation
    - Based on zero-order (ZO) optimization algorithms
  - Search-based
    - Based on choosing a search strategy using a search distribution.

Jacobian-based Dataset Augmentation

Jacobian-based Dataset Augmentation

Practical Black-Box Attacks against Machine Learning

Nicolas Papernot
Pennsylvania State University
ngp5056@cse.psu.edu

Patrick McDaniel
Pennsylvania State University
mcdaniel@cse.psu.edu

Ian Goodfellow*
OpenAI
ian@openai.com

Somesh Jha
University of Wisconsin
jha@cs.wisc.edu

Z. Berkay Celik
Pennsylvania State University
zbc102@cse.psu.edu

Ananthram Swami
US Army Research Laboratory
ananthram.swami.civ@mail.mil

Abstract

The first black-box attack against DNN classifiers for real-world adversaries with no knowledge about the model.

- The only capability of the black-box adversary is to observe **labels** given by the DNN to chosen inputs.

The attack strategy

- **Training a local model** to substitute for the target DNN.
  - Using **inputs synthetically generated** by an adversary and labeled by the target DNN.
- The local substitute is used to craft adversarial examples, and find that they are misclassified by the targeted DNN.

To perform a real-world and properly-blinded evaluation, we attack a DNN hosted by **MetaMind**, an online deep learning API.

Threat Model

In the black-box setting, adversaries do not know internal details of a system to compromise it.

- The adversary has **no information about the structure or parameters** of the target DNN.
- The adversary has **no knowledge of the training data** used to learn the DNN's parameters.
- The adversary does **not have access to any large training dataset**.
- The adversary's only capability is to observe **labels** assigned by the DNN for chosen inputs.

Threat Model

In the black-box setting, adversaries do not know internal details of a system to compromise it.

- The adversary has **no information about the structure or parameters** of the target DNN.
- The adversary has **no knowledge of the training data** used to learn the DNN's parameters.
- The adversary does **not have access to any large training dataset**.
- The adversary's only capability is to observe **labels** assigned by the DNN for chosen inputs.

A data-limited adversary

- Many modern machine learning systems require large and expensive training sets for training.
    - This makes attacks based on training substitute model **unfeasible for adversaries without large labeled datasets**.
- To enable the adversary to train a substitute model without a real labeled dataset
    - The adversary **uses the target DNN as an oracle to construct a synthetic dataset**.

Adversarial Capabilities

The oracle O is the targeted DNN.

- Its name refers to the only capability of the adversary: accessing the label $\tilde{O}(x)$ for any input x by querying oracle O.

The output label $\tilde{O}(x)$ is the index of the class assigned the largest probability by the DNN

$$\tilde{O}(x) = \underset{j \in 0,\ldots,N-1}{argmax} O_j(x)$$

where $O_j(x)$ is the j-th component of the probability vector $O(x)$ output by DNN O, and N is the number of classes.

Accessing labels $\tilde{O}$ produced by the DNN O is the only capability assumed in our threat model.

Adversarial Goal

The adversary wants to produce a minimally altered version of any input x, named **adversarial example**, and denoted x^*, misclassified by oracle O

$$x^* = x + argmin\{z : \tilde{O}(x+z) \neq \tilde{O}(x)\} = x + \delta_x$$
$$\text{such that}: \ \tilde{O}(x^*) \neq \tilde{O}(x)$$

Black-box Attack

Black-box Attack Strategy

1. **Substitute Model Training**: the attacker queries the oracle with **synthetic inputs** selected by a Jacobian based heuristic to build a **model** F **approximating the oracle model** O**'s decision boundaries**.
2. **Adversarial Sample Crafting**: the attacker uses substitute network F to craft adversarial samples, which are then misclassified by oracle O due to the transferability of adversarial samples.

Black-box Attack

Black-box Attack Strategy

1. **Substitute Model Training**: the attacker queries the oracle with **synthetic inputs** selected by a Jacobian based heuristic to build a **model** F **approximating the oracle model** O**'s decision boundaries**.
2. **Adversarial Sample Crafting**: the attacker uses substitute network F to craft adversarial samples, which are then misclassified by oracle O due to the transferability of adversarial samples.

Challenges

- **Select an architecture** for our substitute without knowledge of the targeted oracle's architecture
- **Limit the number of queries** made to the oracle in order to ensure that the approach is tractable.

Generating a Synthetic Dataset

The heuristic used to generate **synthetic training inputs** is based on identifying **directions in which the model's output is varying**, around an initial set of training points.

Generating a Synthetic Dataset

The heuristic used to generate **synthetic training inputs** is based on identifying **directions in which the model's output is varying**, around an initial set of training points.

- These directions are identified with the substitute DNN's **Jacobian matrix** J_F , which is evaluated at several input points x.
- Precisely, the adversary evaluates the sign of the Jacobian matrix dimension corresponding to the **label assigned to input** x **by the oracle**

$$sign(J_F(x)[\tilde{O}(x)]) = sign(\nabla_x F(x)_{\tilde{O}(x)})$$

where $F(x)_i$ is the i-th element of the probability vector $F(x)$ output by substitute model F.

Generating a Synthetic Dataset

The heuristic used to generate **synthetic training inputs** is based on identifying **directions in which the model's output is varying**, around an initial set of training points.

- These directions are identified with the substitute DNN's **Jacobian matrix** J_F , which is evaluated at several input points x.
- Precisely, the adversary evaluates the sign of the Jacobian matrix dimension corresponding to the **label assigned to input** x **by the oracle**

$$sign(J_F(x)[\tilde{O}(x)]) = sign(\nabla_x F(x)_{\tilde{O}(x)})$$

where $F(x)_i$ is the i-th element of the probability vector $F(x)$ output by substitute model F.

The new synthetic training point x' is created as follows

$$x' = x + \lambda.sign(J_F(x)[\tilde{O}(x)]) = x + \lambda.sign(\nabla_x F(x)_{\tilde{O}(x)})$$

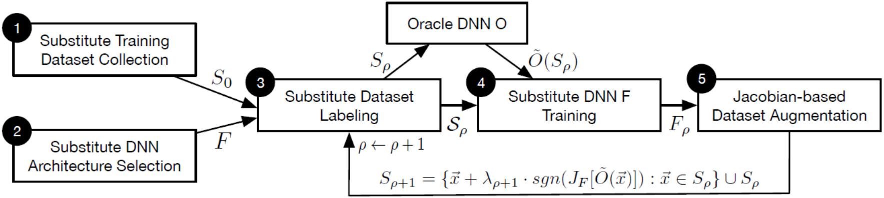
where x is already in the training set, and λ is a parameter of the augmentation.

This technique is called **Jacobian-based Dataset Augmentation**.

Jacobian-based Dataset Augmentation

- **Initial Collection**: The adversary collects a very small set S_0 of inputs representative of the input domain.
- **Architecture Selection**: The adversary selects an architecture to be trained as the substitute model F.
- **Substitute Training**: the adversary iteratively trains more accurate substitute model F_ρ by repeating the following for max_ρ rounds:
  1. **Labeling** S_ρ by oracle O
  2. **Training** F_ρ on S_ρ
  3. Create $S_{\rho+1}$ by **augmenting** S_ρ with more synthetic training points.

$$S_{\rho+1} = \{x + \lambda_{\rho+1}.sign(J_F(x)[\tilde{O}(x)]) : \ x \in S_\rho\} \cup S_\rho$$



Algorithm

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ

Define architecture F
for $\rho \in 0 \; .. \; max_\rho - 1$ **do**
    // Label the substitute training set
    $D \leftarrow \left\{ (\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho \right\}$
    // Train F on D to evaluate parameters θ_F
    $\theta_F \leftarrow \mathrm{train}(F, D)$
    // Perform Jacobian-based dataset augmentation
    $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \mathrm{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
end for
return θ_F

Next Papers

They introduce the *Jacobian-based Dataset Augmentation* (JbDA) technique for generating synthetic samples (row 13). It relies on computing the Jacobian matrices with the current F' evaluated on the already labeled samples in L. Each element $x \in L$ is modified by adding the sign of the Jacobian matrix $\nabla_x \mathcal{L}(F'(x, c_i))$ dimension corresponding to the label assigned to x by F, evaluated with regards to the classification loss $\mathcal{L}$. Thus, the set U is extended with $\{x + \lambda \cdot sign(\nabla_x \mathcal{L}(F'(x, c_i)))\}$, $\forall x \in L$. U has the same

PRADA, IEEE European S&P, 2018

```
# Get augmented inputs
X, Y = X.to(self.device), Y.to(self.device)
delta_i = self.fgsm_untargeted(model_adv, X, Y.argmax(dim=1), device=self.device, epsilon=step_size)
# Get corresponding outputs from blackbox
if self.aug_strategy == 'jbda':
    Y_i = self.blackbox(X + delta_i)
```

Prediction Poisoning, ICLR, 2020

2. *Jacobian Based Dataset Augmentation (JBDA)* (Papernot et al., 2017) uses synthetic data to query the target model. JBDA starts with a small set of in-distribution "seed" examples $\{x_i\}$. The attack iteratively trains the clone model by performing the following steps: (i) Obtain a labeled dataset $\{x_i, y_i\}$ by querying the target model (ii) Train the clone model on the labeled dataset (iii) Augment the dataset with synthetic examples x_i' by perturbing the original input x_i to change the prediction of the clone model by using the Jacobian of the loss function: $x_i' = x_i + \beta sign\left(\nabla_x \mathcal{L}\left(C\left(x_i; \theta_c\right), y_i\right)\right)$.

EDM, ICLR, 2021

generates a synthetic example x', by perturbing it using the jacobian of the clone model's loss function: $x' = x + \lambda sign\left(\nabla_x \mathcal{L}\left(f'\left(x; \theta'\right)\right)\right)$. These synthetic examples are labeled using the predictions of the defender's model $y' = f(x')$ and the labeled synthetic examples thus generated: $\mathcal{D}_{syn} = \{x', y'\}$, are used to augment the adversary's dataset: $\mathcal{D}_{seed} = \mathcal{D}_{seed} \cup \mathcal{D}_{syn}$ and retrain f'.

Adaptive Misinformation, CVPR, 2019

Attack on MetaMind MNIST Model

Initial Substitute Training Sets

- **MNIST subset**: This initial substitute training set is made of 150 samples from the MNIST test set.
- **Handcrafted set**: 100 samples by handwriting 10 digits for each class between 0 and 9 with a laptop trackpad.

Implementation details

- $max_\rho = 6$.
- During each of these 6 rounds, the model is trained for 10 epochs from scratch.
- $\lambda = 0.1$

The Accuracy of the Two Substitute Models

Substitute Epoch	Initial Substitute Training Set from MNIST test set	Handcrafted digits
0	24.86%	18.70%
1	41.37%	19.89%
2	65.38%	29.79%
3	74.86%	36.87%
4	80.36%	40.64%
5	79.18%	56.95%
6	81.20%	67.00%

Figure 4: **Substitute DNN Accuracies:** each column corresponds to an initial substitute training set: 150 MNIST test samples, and handcrafted digits. Accuracy is reported on the unused 9,850 MNIST test samples.

Evaluation

- MNIST test samples are used to generate adversarial examples using FGSM.
- The success rate is the proportion of adversarial samples misclassified by the substitute model.
- The transferability of adversarial samples refers to the oracle misclassification rate of adversarial samples crafted using the substitute DNN.

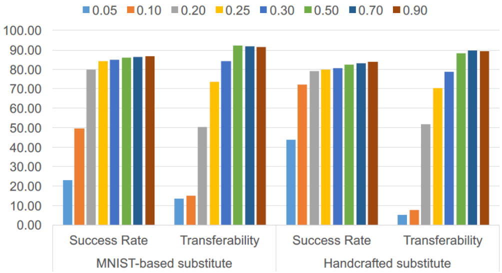


Figure 5: **Success Rate and Transferability of Adversarial Samples for the MetaMind attacks:** performed using MNIST-based and handcrafted substitutes: each bar corresponds to a different perturbation input variation.

Attack Algorithm Calibration

- The choice of **substitute DNN architecture** (number of layers, size, activation function, type) has a **limited impact** on adversarial sample transferability.

Attack Algorithm Calibration

- The choice of **substitute DNN architecture** (number of layers, size, activation function, type) has a **limited impact** on adversarial sample transferability.
- **Increasing the number of epochs**, after the substitute DNN has reached an asymptotic accuracy, does not improve adversarial sample transferability.

Attack Algorithm Calibration

- The choice of **substitute DNN architecture** (number of layers, size, activation function, type) has a **limited impact** on adversarial sample transferability.
- **Increasing the number of epochs**, after the substitute DNN has reached an asymptotic accuracy, does not improve adversarial sample transferability.
- Increasing **step size λ negatively impacts** adversarial sample transferability and does not modify the substitute accuracy by more than 3%.

Attack Algorithm Calibration

- The choice of **substitute DNN architecture** (number of layers, size, activation function, type) has a **limited impact** on adversarial sample transferability.
- **Increasing the number of epochs**, after the substitute DNN has reached an asymptotic accuracy, does not improve adversarial sample transferability.
- Increasing **step size λ negatively impacts** adversarial sample transferability and does not modify the substitute accuracy by more than 3%.
- Having the **step size periodically alternating between positive and negative values improves the quality** of the oracle approximation made by the substitute.

Attack Algorithm Calibration

- The choice of **substitute DNN architecture** (number of layers, size, activation function, type) has a **limited impact** on adversarial sample transferability.
- **Increasing the number of epochs**, after the substitute DNN has reached an asymptotic accuracy, does not improve adversarial sample transferability.
- Increasing **step size λ negatively impacts** adversarial sample transferability and does not modify the substitute accuracy by more than 3%.
- Having the **step size periodically alternating between positive and negative values improves the quality** of the oracle approximation made by the substitute.
- **Reducing Oracle Querying**: randomly select κ samples from a list of samples. The adversary after σ iterations selects κ new inputs for Jacobian-based dataset augmentation.

ZOO: Zeroth Order Optimization Based Black-box Attacks

ZOO: Zeroth Order Optimization Based Black-box Attacks

ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models

Pin-Yu Chen*
AI Foundations Group
IBM T. J. Watson Research Center
Yorktown Heights, NY
pin-yu.chen@ibm.com

Huan Zhang*†
University of California, Davis
Davis, CA
ecezhang@ucdavis.edu

Yash Sharma
IBM T. J. Watson Research Center
Yorktown Heights, NY
Yash.Sharma3@ibm.com

Jinfeng Yi
AI Foundations Group
IBM T. J. Watson Research Center
Yorktown Heights, NY
jinfengy@us.ibm.com

Cho-Jui Hsieh
University of California, Davis
Davis, CA
chohsieh@ucdavis.edu

Abstract

Throughout this paper,we consider a practical **black-box attack** setting where one can access the input and output of a DNN but not the internal configurations.

- In this setting, back propagation for gradient computation of the targeted DNN is prohibited.

We propose **Zeroth Order Optimization (ZOO)** based attacks to directly **estimate the gradients** of the targeted DNN for generating adversarial examples.

Black-box Attack Using Zeroth Order Optimization

Zeroth order methods are **derivative-free optimization** methods, where only the zeroth order oracle (the objective function value $f(x)$ at any x) is needed during optimization process.

- By evaluating the objective function values at **two very close points** $f(x + hv)$ and $f(x - hv)$ with a small h, **a proper gradient along the direction vector** v **can be estimated**.
- Then, classical optimization algorithms like gradient descent or coordinate descent can be applied using the estimated gradients.
  - **White-box attack using estimated gradients**

Notation for deep neural networks

Model $F(x)$ takes an image $x \in \mathbb{R}^p$ as an input and outputs a vector $F(x) \in [0,1]^K$ of confidence scores for each class, where K is the number of classes.

- The k-th entry $[F(x)]_k \in [0,1]$specifies the probability of classifying x as class k, and $\sum_{k=1}^{K}[F(x)]_k = 1$.

Formulation of C&W attack

Our black-box attack is inspired by the formulation of the C&W attack. The C&W attack finds the adversarial example x by solving the following optimization problem:

$$\begin{aligned} \underset{\delta}{\text{Minimize}} \quad & \|\delta\|_2^2 + c.f(x,t) \\ \text{subject to:} \quad & x = x_0 + \delta \\ & x \in [0,1]^p \end{aligned}$$

where x_0 is clean data, t is the target class, c is a regularization parameter, and $f(x,t)$ is defined as follows

$$f(x,t) = max\{\underset{i \neq t}{max}[Z(x)]_i - [Z(x)]_t, -\kappa\}$$

where $Z(x) \in \mathbb{R}^K$ is the logit layer representation (logits).

Black-box Attack via Zeroth Order Stochastic Coordinate Descent

We amend C&W attack to the black-box setting by proposing the following approaches

- **Modify the loss function** $f(x, t)$ such that it **only depends on the output** F of a DNN and the target class label t.
- Solve the optimization problem via zeroth order optimization.
  - Compute an approximate gradient using a **Finite Difference Method** instead of actual back propagation on the targeted DNN

Loss function $f(x,t)$ based on f

Inspired by $C\&W$ attack, we propose a new loss function based on the output F of a DNN, which is defined as

$$f(x,t) = max\{\underset{i \neq t}{max} \ \log[F(x)]_i - \log[F(x)]_t, -\kappa\}$$

where $\kappa \geq 0$. We find that the **log operator** is essential to our black-box attack.

For **untargeted attacks**, an adversarial attack is successful when x is classified as any class other than the original class label t_0. A similar loss function can be used

$$f(x) = max\{\log[F(x)]_{t_0} - \underset{i \neq t_0}{max} \ \log[F(x)]_i, -\kappa\}$$

where t_0 is the original class label for x.

Zeroth Order Optimization on the Loss Function

We discuss our optimization techniques **for any general function** f used for attacks. We use the **Symmetric Difference Quotient** to estimate the gradient $\hat{g}_i = \frac{\partial f(x)}{\partial x_i}$

$$\hat{g}_i = \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h},$$

where h is a small constant ($h = 0.0001$) and e_i is a standard basis vector with only the i-th component as 1.

Zeroth Order Optimization on the Loss Function

We discuss our optimization techniques **for any general function** f used for attacks. We use the **Symmetric Difference Quotient** to estimate the gradient $\hat{g}_i = \frac{\partial f(x)}{\partial x_i}$

$$\hat{g}_i = \frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + he_i) - f(x - he_i)}{2h},$$

where h is a small constant ($h = 0.0001$) and e_i is a standard basis vector with only the i-th component as 1.

For any $x \in \mathbb{R}^p$, we need to **evaluate the objective function $2p$ times** to estimate gradients of all p coordinates.

- This naive solution is too expensive in practice.
- Even for an input image size of $64 \times 64 \times 3$, one full gradient descent step requires $24,576$ evaluations, and typically hundreds of iterations may be needed until convergence.

Therefore, using stochastic **gradient descent for minimizing objective function is too expensive**.

Stochastic Coordinate Descent

At each iteration, **one variable (coordinate)** is chosen randomly and is updated by approximately minimizing the objective function along that coordinate (Algorithm 1).

- δ^* is approximated by $-\eta \hat{g}_i$ (where η is the learning rate).
- In our implementation, we estimate $B = 128$ pixels' gradients per iteration, and then update B coordinates in a single iteration.

The attack uses zeroth order coordinate ADAM.

Algorithm 1 Stochastic Coordinate Descent

while not converged **do**
  Randomly pick a coordinate $i \in \{1, \ldots, p\}$
  Compute an update δ^* by approximately minimizing

$$\arg\min_{\delta} f(\mathbf{x} + \delta \mathbf{e}_i)$$

  Update $\mathbf{x}_i \leftarrow \mathbf{x}_i + \delta^*$
end while

Reducing Attack Cost

For networks with a **large input size** p, optimizing over $\mathbb{R}^p$ (we call it attack-space) using zeroth order methods can be quite slow because we need to estimate a **large number of gradients**.

Reducing Attack Cost

For networks with a **large input size** p, optimizing over $\mathbb{R}^p$ (we call it attack-space) using zeroth order methods can be quite slow because we need to estimate a **large number of gradients**.

Some methods to accelerate the attack process

- **Attack-space dimension reduction**
  - Reduces the dimension of attack-space from p to m ($m < p$).
  - Optimize δ over $\mathbb{R}^m$
  - Upscale δ from $\mathbb{R}^m$ to $\mathbb{R}^p$ in order to generate adversarial example by adding δ to $x \in \mathbb{R}^p$

Reducing Attack Cost

For networks with a **large input size** p, optimizing over $\mathbb{R}^p$ (we call it attack-space) using zeroth order methods can be quite slow because we need to estimate a **large number of gradients**.

Some methods to accelerate the attack process

- **Attack-space dimension reduction**
  - Reduces the dimension of attack-space from p to m ($m < p$).
  - Optimize δ over $\mathbb{R}^m$
  - Upscale δ from $\mathbb{R}^m$ to $\mathbb{R}^p$ in order to generate adversarial example by adding δ to $x \in \mathbb{R}^p$
- **Hierarchical attack**
  - For large images and difficult attacks, we propose to use a hierarchical attack scheme, where we use a series of transformations with dimensions $m_1, m_2, \cdots$ ($m_2 > m_1$) to gradually increase m during the optimization process.

Reducing Attack Cost

For networks with a **large input size** p, optimizing over $\mathbb{R}^p$ (we call it attack-space) using zeroth order methods can be quite slow because we need to estimate a **large number of gradients**.

Some methods to accelerate the attack process

- **Attack-space dimension reduction**
  - Reduces the dimension of attack-space from p to m ($m < p$).
  - Optimize δ over $\mathbb{R}^m$
  - Upscale δ from $\mathbb{R}^m$ to $\mathbb{R}^p$ in order to generate adversarial example by adding δ to $x \in \mathbb{R}^p$
- **Hierarchical attack**
  - For large images and difficult attacks, we propose to use a hierarchical attack scheme, where we use a series of transformations with dimensions $m_1, m_2, \cdots$ ($m_2 > m_1$) to gradually increase m during the optimization process.
- **Optimize the important pixels first**
  - Since estimating gradient for each pixel is expensive in the black-box setting, we propose to selectively update pixels by using importance sampling.
  - We propose to divide the image into 8×8 regions, and assign sampling probabilities according to how large the pixel values change in that region.

Evaluation

We compare ZOO with

- Carlini & Wagner's (C&W) white-box attack
- The substitute model based black-box attack (JBDA attack to create substitute model)

Setup

- Batch size of $B = 128$
  - Evaluate 128 gradients and update 128 coordinates per iteration.
- Set $\kappa = 0$
- Binary search up to 9 times to find the best c in C&W attack.
- We run 3000 iterations for MNIST and 1000 iterations for CIFAR10 (gradient descent iteration, each iteration update 128 coordinates)
  - $3000 \times 128 \times 2 \times 9 = 6,912,000$ queries for single adversarial example on MNIST model
  - $1000 \times 128 \times 2 \times 9 = 2,304,000$ queries for single adversarial example on CIFAR10 model
- Since the image size of MNIST and CIFAR10 is small, we do not reduce the dimension of attack-space or use hierarchical attack and importance sampling.

Evaluation

Table 1: MNIST and CIFAR10 attack comparison: ZOO attains comparable success rate and L_2 distortion as the white-box C& W attack, and significantly outperforms the black-box substitute model attacks using FGSM (L_∞ attack) and the C&W attack [35] The numbers in parentheses in Avg. Time field is the total time for training the substitute model. For FGSM we do not compare its L_2 with other methods because it is an L_∞ attack.

	MNIST					
	Untargeted			Targeted		
	Success Rate	Avg. L_2	Avg. Time (per attack)	Success Rate	Avg. L_2	Avg. Time (per attack)
White-box (C&W)	100 %	1.48066	0.48 min	100 %	2.00661	0.53 min
Black-box (Substitute Model + FGSM)	40.6 %	-	0.002 sec (+ 6.16 min)	7.48 %	-	0.002 sec (+ 6.16 min)
Black-box (Substitute Model + C&W)	33.3 %	3.6111	0.76 min (+ 6.16 min)	26.74 %	5.272	0.80 min (+ 6.16 min)
Proposed black-box (ZOO-ADAM)	100 %	1.49550	1.38 min	98.9 %	1.987068	1.62 min
Proposed black-box (ZOO-Newton)	100 %	1.51502	2.75 min	98.9 %	2.057264	2.06 min
	CIFAR10					
	Untargeted			Targeted		
	Success Rate	Avg. L_2	Avg. Time (per attack)	Success Rate	Avg. L_2	Avg. Time (per attack)
White-box (C&W)	100 %	0.17980	0.20 min	100 %	0.37974	0.16 min
Black-box (Substitute Model + FGSM)	76.1 %	-	0.005 sec (+ 7.81 min)	11.48 %	-	0.005 sec (+ 7.81 min)
Black-box (Substitute Model + C&W)	25.3 %	2.9708	0.47 min (+ 7.81 min)	5.3 %	5.7439	0.49 min (+ 7.81 min)
Proposed Black-box (ZOO-ADAM)	100 %	0.19973	3.43 min	96.8 %	0.39879	3.95 min
Proposed Black-box (ZOO-Newton)	100 %	0.23554	4.41 min	97.0 %	0.54226	4.40 min

Evaluation on ImageNet

Attack setup on ImageNet

- Attack-space of only $32 \times 32 \times 3$
- Fix $c = 10$
- 1500 iterations of gradient descent, which takes about 20 minutes per attack
- $1500 \times 128 = 192000$ gradients are evaluated, less than the total number of pixels ($299 \times 299 \times 3 = 268, 203$) of the input image
  - $1500 \times 128 \times 2 = 384000$ queries for single adversarial example on ImageNet model

Table 2: Untargeted ImageNet attacks comparison. Substitute model based attack cannot easily scale to ImageNet.

	Success Rate	Avg. L_2
White-box (C&W)	100 %	0.37310
Proposed black-box (ZOO-ADAM)	88.9 %	1.19916
Black-box (Substitute Model)	N.A.	N.A.