M2177.0043 Introduction to Deep Learning

Lecture 15: Adversarial attack and defense[1]

Hyun Oh Song[1]

[1]Dept. of Computer Science and Engineering, Seoul National University

May 12, 2020

[1]Many slides and figures adapted Justin Johnson

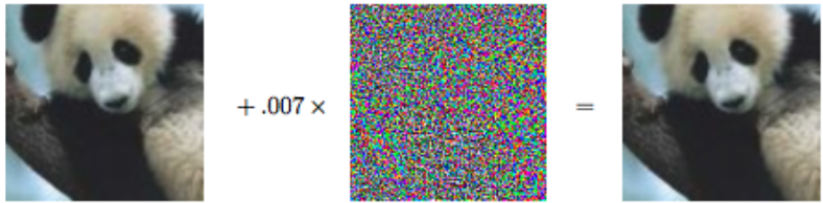
Last time

- Autoregressive generative models: PixelRNN/CNN
- Generative adversarial networks

Outline

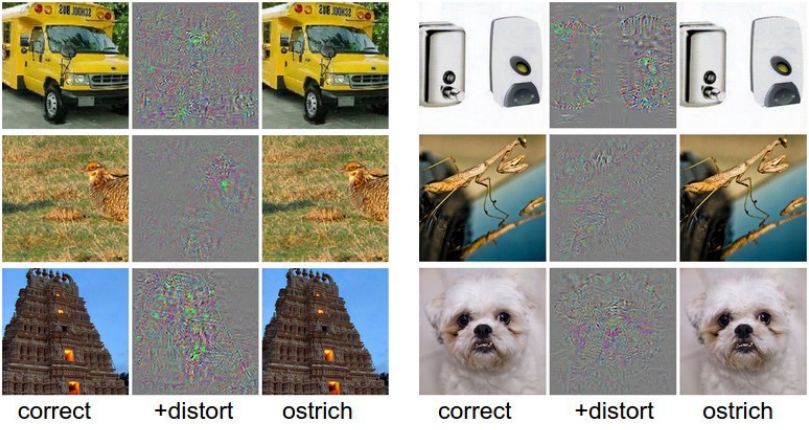
What are adversarial examples?

- Perturbed images that look similar (for human) to original images but can easily fool classifiers.



Ian Goodfellow and Jonathon Shlens and Christian Szegedy, *Explaining and Harnessing Adversarial Examples, 2015*

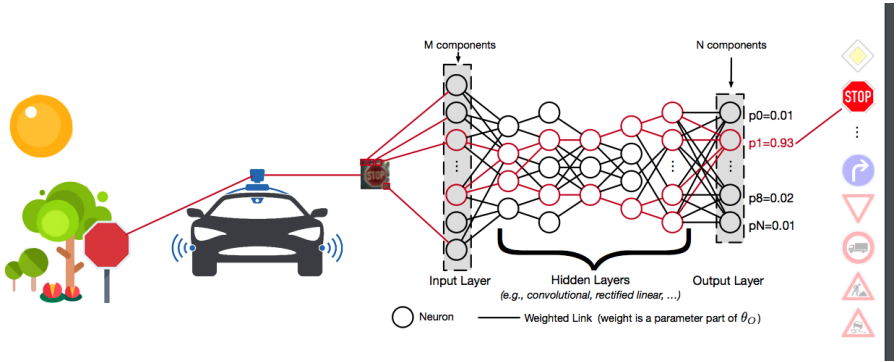
Adversarial Examples



Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus, *Intriguing properties of neural networks*, 2014

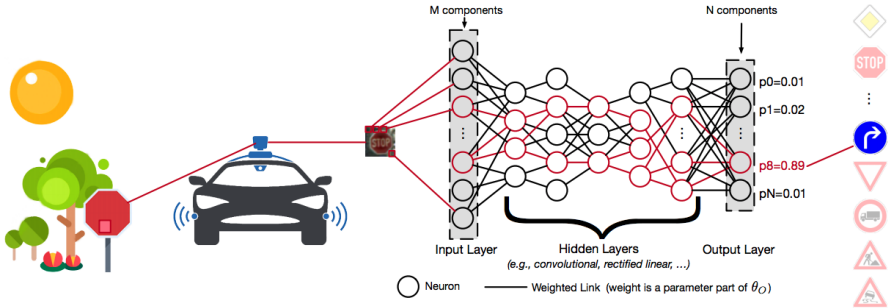
Why are they dangerous?

- Adversarial perturbation can fool the classifier to mispredict with high confidence



Images from Nicolas Papernot's slide: Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks

- Adversarial perturbation can fool the classifier to mispredict with high confidence



Outline

Crafting adversarial examples: Fast Gradient Sign Method

- During training, the classifier uses a loss function to minimize model prediction errors
- After training, attacker uses loss function to maximize model prediction error

- Let $J(x, y, \theta)$ be the loss function that used to train the classifier, $x \in \mathbb{R}^k$ the input to the model, y the label associated with x, and θ the parameters of the model.

- Maximize the loss function $J(x', y, \theta) = J(x + n, y, \theta)$ by adding perturbation to x under ℓ_∞ constraint: $||x' - x||_\infty \leqslant \epsilon$, where ϵ is small enough to be imperceptible by human.

Ian Goodfellow and Jonathon Shlens and Christian Szegedy, *Explaining and Harnessing Adversarial Examples, 2015*

FGSM derivation

- Apply first order Taylor's expansion

$$\underset{x'}{\text{maximize}}\, J(x', y, \theta) = J(x + n, y, \theta) \text{ subject to } ||x' - x||_\infty = ||n||_\infty \leqslant \epsilon,$$
$$\text{where } J(x + n, y, \theta) \simeq J(x, y, \theta) + n^T \nabla_x (J(x, y, \theta)),\ ||n||_\infty \leqslant \epsilon$$

- Upperbound the first order term

$$\begin{aligned} n^T \nabla_x J(x, y, \theta) &\leqslant \|n\|_\infty \|\nabla_x J(x, y, \theta)\|_1 \qquad \text{(Holder's inequality)} \\ &\leqslant \epsilon\ \|\nabla_x J(x, y, \theta)\|_1 \\ &= \epsilon(|\nabla_{x_1} J(x, y, \theta)| + ... + |\nabla_{x_k} J(x, y, \theta)|) \qquad (1) \end{aligned}$$

- if we set $n = \epsilon\ sign(\nabla_x(J(x, y, \theta))) = \epsilon \left(\frac{\nabla_{x_1} J}{|\nabla_{x_1} J|}, \frac{\nabla_{x_2} J}{|\nabla_{x_2} J|}, ..., \frac{\nabla_{x_k} J}{|\nabla_{x_k} J|} \right)$
- then, $n^T \nabla_x J(x, y, \theta) = \epsilon(|\nabla_{x_1} J(x, y, \theta)| + ... + |\nabla_{x_k} J(x, y, \theta)|)$ achieves the upperbound in Equation (1)
- Therefore $x' = x + \epsilon sign(\nabla_x J(x, y, \theta))$ achieves the maximum loss value within the perturbation bound under the first order approximation.

Iterative Projected Gradient Method

- A variant of the fast gradient sign method, where instead of taking a single step size ϵ, smaller steps α are taken repeatedly
- The result with pixel-wise perturbations is clipped by ϵ for each iteration (recall the ℓ_∞ projection operator)
- Such iteration is then repeated for several times. The update process is:

$$x^{t+1} = \prod_{x+\epsilon}(x^t + \alpha sign(\nabla_x J(x^t, y, \theta))),\ x^0 = x$$

Madry, Aleksander and Makelov, Aleksandar and Schmidt, Ludwig and Tsipras, Dimitris and Vladu, Adrian, *Towards deep learning models resistant to adversarial attacks, 2017*

Least Likely Class Method

- The variant of iterative FGSM using y_{LL}, the least likely class, instead of the ground truth label y associated with x.

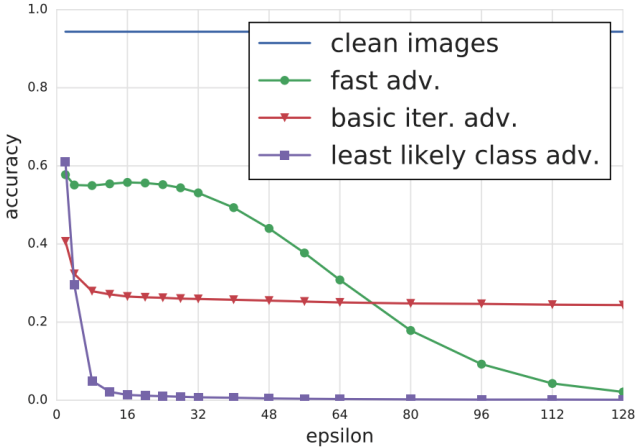
$$y_{LL} = \underset{\widehat{y}}{\operatorname{argmin}}\, P(\widehat{y} \mid x)$$

$$x^{t+1} = \prod_{x+\epsilon}(x^t - \alpha sign(\nabla_x J(x^t, y_{LL}, \theta))),\ x^0 = x$$

- For a well-trained classifier, the least-likely class is usually highly dissimilar from the true class, so this attack method results in more interesting mistakes, such as mistaking a dog for an airplane.

Kurakin, Alexey and Goodfellow, Ian and Bengio, Samy, *Adversarial machine learning at scale, 2017*

Comparison



- Accuracy of Inception v3 under attack by different adversarial methods
- α set to 1.0

Kurakin, Alexey and Goodfellow, Ian and Bengio, Samy, *ADVERSARIAL EXAMPLES IN THE PHYSICAL WORLD, 2017*

Carlini-Wagner Attack

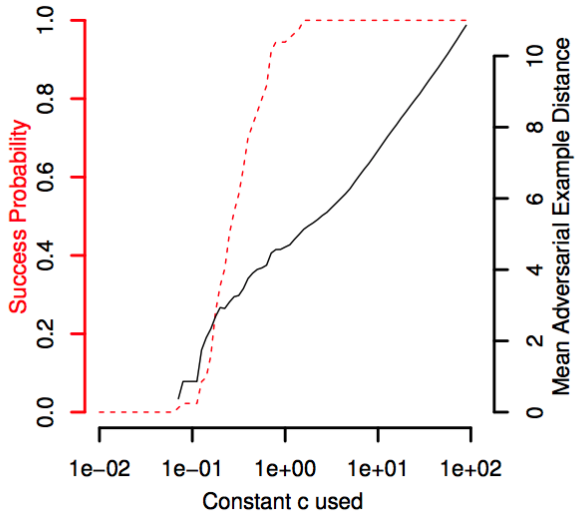
- Carlini *et al.* suggest a neural network which generates adversarial examples that have low distortion in ℓ_2 metric.
- Given x with a chosen target class t an attack example x' is found by optimizing the following

$$\underset{x'}{\text{minimize}} \, \|x' - x\|_2^2 + c \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -k)$$

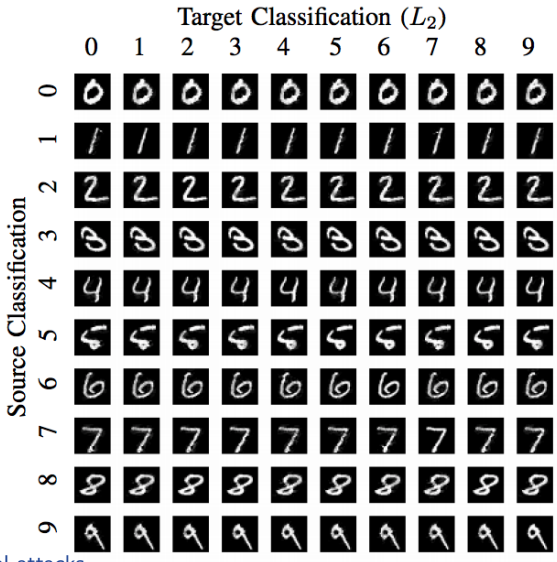
- c is a hyper-parameter that balances the two terms. $Z(x')$ is the logit vector of classifier network and k is a hyper-parameter called confidence.
- Higher confidence encourages the attack to search for adversarial examples that are stronger in classification confidence, but it has larger perturbation.

Carlini, Nicholas and Wagner, David, *Towards evaluating the robustness of neural networks*, 2017

Carlini-Wagner Attack



Carlini-Wagner Attack



Generating Adversarial Examples with Adversarial Networks

- The adversarial loss:

$$L_{GAN} = \mathbb{E}_x \log \mathbf{D}(x) + \mathbb{E}_x \log(1 - \mathbf{D}(x + \mathbf{G}(x))$$

The discriminator $\mathbf{D}$ aims to distinguish the perturbed data $x + \mathbf{G}$ from the original input x.

- The loss for fooling the target model f:

$$L_{adv}^{f} = \mathbb{E}_x \; l_f(x + \mathbf{G}(x), t),$$

where t is the target class and l_f denotes the loss function used to train the original model f

- Additionally, to bound the magnitude of the perturbation :

$$L_{hinge} = \mathbb{E}_x \max(0, ||\mathbf{G}(\mathbf{x})||_2 - c)$$

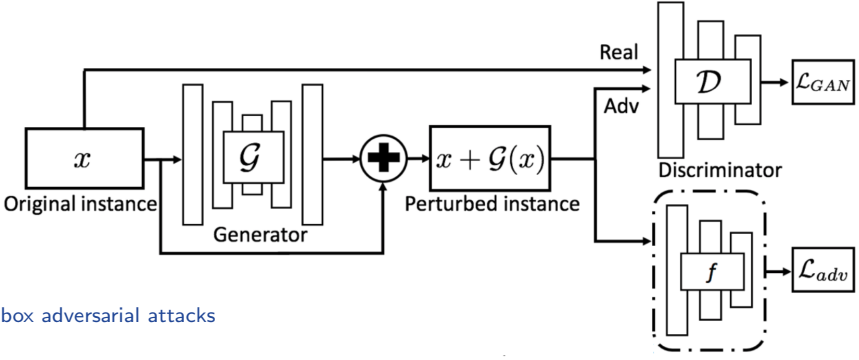
Xiao et al., *Generating Adversarial Examples with Adversarial Networks , 2018*

Generating Adversarial Examples with Adversarial Networks

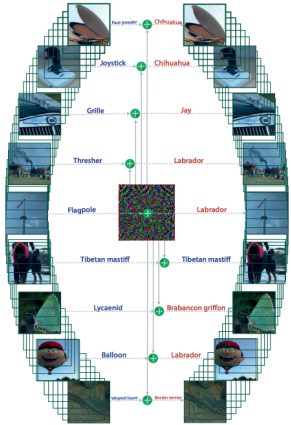
- Full objective:

$$L = L_{adv}^{f} + \alpha L_{GAN} + \beta L_{hinge}$$

- Obtain $\mathbf{G}$ and $\mathbf{D}$ by solving the minmax game $\text{argmin}_G \max_D L$



Universal Adversarial Perturbations



- Goal: Find a single universal perturbation image that can fool the network regardless of the input image

Moosavi-Dezfooli, Seyed-Mohsen and Fawzi, Alhussein and Fawzi, Omar and Frossard, Pascal, *Universal adversarial perturbations, 2017*

Outline

Notation

- Suppose that we have a classifier $C(x)$ with corresponding loss function $\ell(x, y)$, where x is an original image and y is its corresponding label.
- Let y_{adv} be a target class and ϵ be the size of adversarial perturbation.

Formal definition

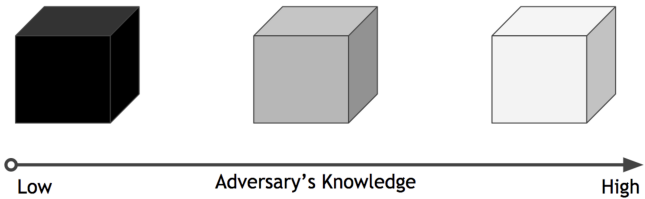
- x_{adv} is called an adversarial example of x if

$$C(x_{adv}) \neq y \quad \text{and} \quad \|x_{adv} - x\|_p \leqslant \epsilon \quad \text{(untargeted)}$$
$$C(x_{adv}) = y_{adv} \quad \text{and} \quad \|x_{adv} - x\|_p \leqslant \epsilon \quad \text{(targeted)}$$

- It can be reformulated as the following constrained optimization problem.

$$\underset{x_{adv}}{\text{maximize}} \ \ell(x_{adv}, y) \ \text{ subject to } \ \|x_{adv} - x\|_p \leqslant \epsilon \quad \text{(untargeted)}$$
$$\underset{x_{adv}}{\text{maximize}} \ -\ell(x_{adv}, y_{adv}) \ \text{ subject to } \ \|x_{adv} - x\|_p \leqslant \epsilon \quad \text{(targeted)}$$

Threat model



- **White-box attack**
  - Adversary can access to the model parameters and the corresponding loss gradient with respect to an input.
- **Black-box attack**
  - Adversary can query an input to the model and receive the class prediction scores, but does not have access to the model parameters.

White-box attacks

Fast Gradient Sign Method[2]

- Compute the gradient of the loss function according to the input pixels. Perturbation is the signs of these derivatives multiplied by ϵ.

$$x_{adv} = x + \epsilon sign(\nabla_x \ell(x, y))$$

Projected Gradient Descent Method[3]

- A variant of the fast gradient sign method, where instead of taking a single step, multiple steps with smaller step size α are taken.

$$x^{t+1} = \Pi_{B_\epsilon(x)}(x^t + \alpha sign(\nabla_x \ell(x^t, y))$$

[2] Ian Goodfellow et al., Explaining and Harnessing Adversarial Examples, 2015
[3] Madry et al., Towards deep learning models resistent to adversarial attack, 2017

Black-box attacks

- Black-box attack is **much more challenging** than white-box attack since we cannot access to the gradient information of the target network.
- Still, black-box attack is a more **realistic setting** since commercial classifiers like Clarifai usually offer only prediction scores.

Black-box attacks with gradient estimation

Recent methods mostly aim at **estimating the true gradient signal** based on the input queries.

- **ZOO**[4] computes the coordinate-wise numerical gradients by querying for central difference values and apply coordinate descent method.

$$\widehat{g}_i \coloneqq \frac{\partial \ell(x)}{\partial x_i} \approx \frac{\ell(x + \sigma e_i) - \ell(x - \sigma e_i)}{2\sigma}$$

- **NES**[5] and **Bandits**[6] compute the vector-wise gradient estimate with randomly sampled vectors $\{u_i\}$ by $\frac{1}{\sigma n}\sum_i^n (\ell(x + \sigma u_i) - \ell(x - \sigma u_i))u_i$ and apply projected gradient descent.

[4] Chen, et al., Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models, 2017

[5] Ilyas, et al., Black-box adversarial attacks with limited queries and information, 2018

[6] Ilyas, et al., Prior convictions: Black-box adversarial attacks with bandits and priors, 2018

Black-box attacks with gradient estimation

- However, these black-box attacks are very susceptible to the choice of **hyperparameters** such as the learning rate, decay rates, and the update rule.
- If we devise an attack algorithm which does not require estimating the gradient, it would be free of the update hyperparameters and thus be more applicable.

Objective

- We focus on **black-box** attacks under ℓ_∞ **constraint** with limited query access.
- We aim to propose a new attack method under above setting with higher performance and query efficiency.

Motivation

- White-box attacks find an adversarial example by deriving the following first order Taylor approximation of the loss function.

$$\ell(x_{adv}, y) \approx \ell(x, y) + (x_{adv} - x)^\intercal \nabla_x \ell(x, y)$$

- Then, the optimization problem becomes a linear program (LP).

$$\underset{\|x_{adv} - x\|_\infty \leqslant \epsilon}{\text{maximize}} \ \ell(x_{adv}, y) \implies \underset{\|x_{adv} - x\|_\infty \leqslant \epsilon}{\text{maximize}} \ x_{adv}^\intercal \nabla_x \ell(x, y)$$

- Since the feasible set is bounded, the solution of the LP is attained at an extreme point of the feasible set[7] and we can characterize that an optimal solution will be attained at a vertex of the ℓ_∞ ball.

[7] Schrijver, Theory of linear and integer programming, 1986

Motivation

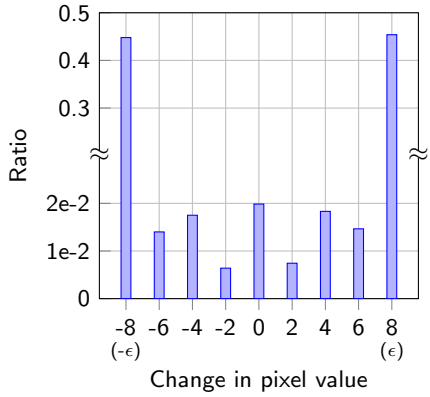


Figure: Distribution of adversarial noise with white box PGD attack on Cifar-10 dataset with wide Resnet w32-10 adversarially trained network at ℓ_∞ ball radius $\epsilon = 8$ in [0, 255] scale.

Motivation

- This characterization motivates us to consider a **discrete surrogate** to the problem as follows.

$$\underset{\|x_{adv}-x\|_\infty \leqslant \epsilon}{\text{maximize}} \ \ell(x_{adv}, y) \implies \underset{x_{adv}-x \in \{-\epsilon,\epsilon\}^d}{\text{maximize}} \ \ell(x_{adv}, y)$$

where d denotes the number of pixels in the image x.

Problem formulation

- Equivalently, the discrete problem can be reformulated as the following **set maximization problem**.

$$\underset{\mathcal{S}\subseteq\mathcal{V}}{\text{maximize}} \quad \left\{ F(\mathcal{S}) \triangleq \ell\left(x + \epsilon \sum_{i\in\mathcal{S}} e_i - \epsilon \sum_{i\notin\mathcal{S}} e_i, y \right) \right\}$$

  where $\mathcal{V}$ denotes the set of all pixel locations, $\mathcal{S}$ denotes the set of selected pixels with $+\epsilon$ perturbations, and $\mathcal{V} \setminus \mathcal{S}$ indicates the set of remaining pixels with $-\epsilon$ perturbations.

- Now, our goal is to find $\mathcal{S}$ that maximizes the objective set function.

Submodular function

- In general, maximizing a set function is NP-hard.
- However, many set functions that arise in machine learning problem often exhibit **submodularity**.

Submodular function

Definition 1

For a set function $F : 2^{\mathcal{V}} \to \mathbb{R}$, $\mathcal{S} \subseteq \mathcal{V}$, and $e \in \mathcal{V} \setminus \mathcal{S}$, the marginal gain of F at $\mathcal{S}$ with respect to e is defined by

$$\Delta(e \mid \mathcal{S}) := F(\mathcal{S} \cup \{e\}) - F(\mathcal{S})$$

Definition 2

A function $F : 2^{\mathcal{V}} \to \mathbb{R}$ is submodular if for every $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{V}$ and $e \in \mathcal{V} \setminus \mathcal{B}$, it holds that

$$\Delta(e \mid \mathcal{A}) \geqslant \Delta(e \mid \mathcal{B})$$

Greedy algorithm for submodular maximization

- Submodular functions have a diminishing returns property where the marginal gain diminishes as the set size increases.
- Using this property, we can efficiently compute an approximate optimal solution of submodular function with **greedy-style algorithms**.
- These greedy-style algorithms provide theoretical guarantees with approximation bounds.
  - $(1 - \frac{1}{e})$-approximation for monotone submodular function

Local search algorithm

- For non-monotone submodular functions, **local search algorithm**[8] performs better than standard greedy insertion algorithms, giving a $\frac{1}{3}$-approximation bound.
- The algorithm alternates between greedily inserting an element and removing an element while the marginal gain is strictly positive.

input Objective set function F, Working set $\mathcal{S}$, Ground set $\mathcal{V}$
1: **for** $t = 1, \ldots,$ MAXITER **do**
2: Greedily insert elements of $\mathcal{V}$ into $\mathcal{S}$ while the marginal gain is strictly positive
3: Greedily delete elements from $\mathcal{S}$ while the marginal gain is strictly positive
4: **end for**
output $\underset{\mathcal{A} \in \{\mathcal{S}, \mathcal{V} \setminus \mathcal{S}\}}{\operatorname{argmax}} F(\mathcal{A})$

Algorithm 1: Local Search algorithm

[8] Feige, et al., Maximizing non-monotone submodular functions, 2011

Local search optimization on approximate submodular function

- Unfortunately, our objective function may not be exactly submodular.
- However, we empirically found that running the local search algorithm on our objective function leads to a good performance comparable to well-known **white-box** attacks in some settings.
- Also, we proved that the algorithm also works to a substantial extent when the submodularity is not severely deteriorated.

Local search optimization on approximate submodular function

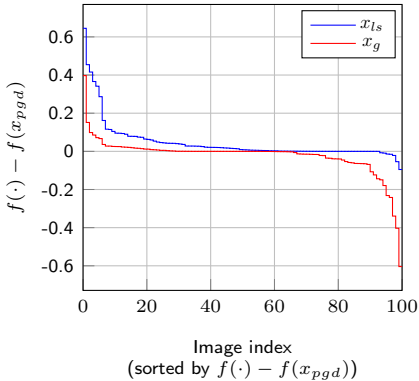


Figure: $f(\cdot) - f(x_{pgd})$ values on random 100 samples on Cifar-10, where x_{ls}, x_g and x_{pgd} each denotes image perturbed by local search, greedy insertion and PGD method.

Local search optimization on approximate submodular function

Theorem 3

Let $\mathcal{C}$ be an optimal solution for a function F and $\mathcal{S}$ be the solution obtained by the local search algorithm. Then,

$$2F(\mathcal{S}) + F(\mathcal{V} \setminus \mathcal{S}) \geqslant F(\mathcal{C}) + \xi \lambda_F(\mathcal{V}, 2),$$

where

$$\xi = \binom{|\mathcal{S} \setminus \mathcal{C}|}{2} + \binom{|\mathcal{C} \setminus \mathcal{S}|}{2} + |\overline{\mathcal{S} \cup \mathcal{C}}| \cdot |\mathcal{S}| + |\mathcal{C} \setminus \mathcal{S}| \cdot |\mathcal{S} \cap \mathcal{C}|$$

and $\lambda_F(\cdot, \cdot)$ is the submodularity index[9] of F, which is a measure of the degree of submodularity.

[9] Zhou and Spanos, Casual meets submodular: Subset selection with directed information, 2016

Local search optimization on approximate submodular function

Corollary 4

If $F(\mathcal{C}) + \xi\lambda_F(\mathcal{V}, 2) \geqslant 0$, *then one of the following holds.*

1. $F(\mathcal{S}) \geqslant \frac{1}{3}(F(\mathcal{C}) + \xi\lambda_F(\mathcal{V}, 2))$
2. $F(\mathcal{V} \setminus \mathcal{S}) \geqslant \frac{1}{3}(F(\mathcal{C}) + \xi\lambda_F(\mathcal{V}, 2))$

Black-box adversarial attack via local search optimization

- Applying the local search algorithm on our objective set function, we obtain a set $\mathcal{S}$ of pixels to perturb the input x with $+\epsilon$ and its complement $\mathcal{V} \setminus \mathcal{S}$ to perturb with $-\epsilon$.
- Finally, the perturbed image is computed as

$$x_{adv} \triangleq x + \epsilon \sum_{i \in \mathcal{S}} e_i - \epsilon \sum_{i \notin \mathcal{S}} e_i$$

Adversarial examples of our algorithm

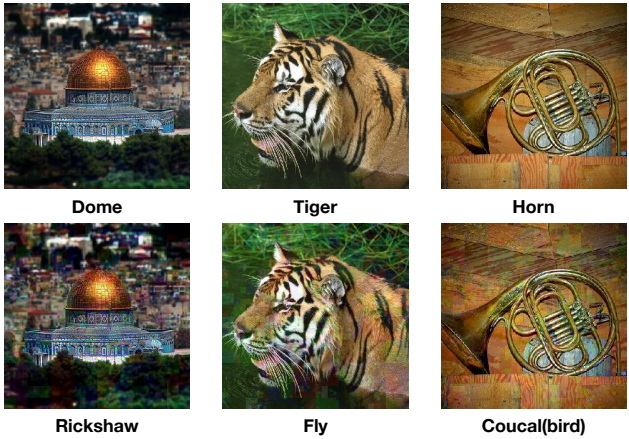


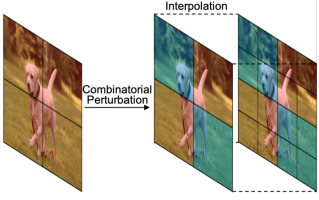
Figure: Adversarial examples from ImageNet in the targeted setting. The top row shows the original images and the bottom row shows the corresponding perturbed images from our method.

Hierarchical approach for query efficiency

- Running the local search algorithm on the set of all pixels could be query inefficient.
- To address this problem, we take a hierarchical approach, performing the local search on a coarse grid and use the results in the subsequent round on finer grid.

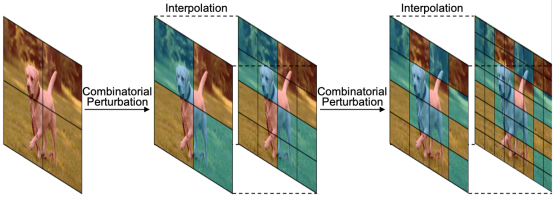
Hierarchical approach for query efficiency

- Running the local search algorithm on the set of all pixels could be query inefficient.
- To address this problem, we take a hierarchical approach, performing the local search on a coarse grid and use the results in the subsequent round on finer grid.



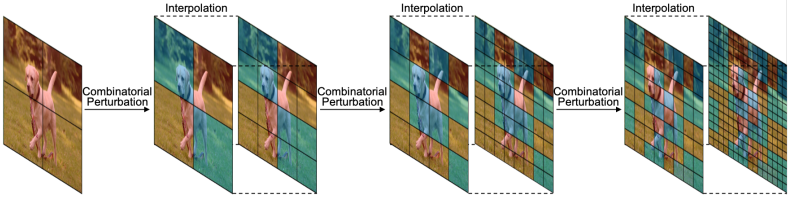
Hierarchical approach for query efficiency

- Running the local search algorithm on the set of all pixels could be query inefficient.
- To address this problem, we take a hierarchical approach, performing the local search on a coarse grid and use the results in the subsequent round on finer grid.



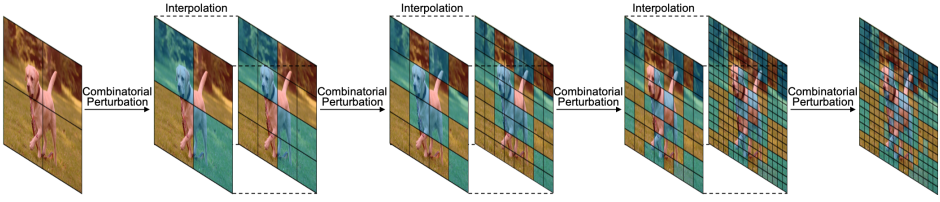
Hierarchical approach for query efficiency

- Running the local search algorithm on the set of all pixels could be query inefficient.
- To address this problem, we take a hierarchical approach, performing the local search on a coarse grid and use the results in the subsequent round on finer grid.



Hierarchical approach for query efficiency

- Running the local search algorithm on the set of all pixels could be query inefficient.
- To address this problem, we take a hierarchical approach, performing the local search on a coarse grid and use the results in the subsequent round on finer grid.



Pseudocode

input Objective set function F, Block size k, Ground set $\mathcal{V}$ of size $|\mathcal{V}| = h/k \times w/k \times c$ where the image size is $h \times w \times c$
initialize Working set $\mathcal{S} = \emptyset$
repeat
  Run Local Search Algorithm on $\mathcal{S}$ and $\mathcal{V}$ using Algorithm 1
  $\mathcal{S} \leftarrow \textsc{LocalSearch}(F, \mathcal{S}, \mathcal{V})$
  if $k > 1$ **then**
    Split the blocks in $\mathcal{S}$ and $\mathcal{V}$ into finer blocks
    $\mathcal{S} \leftarrow \textsc{SplitBlock}(\mathcal{S}, k)$, $\mathcal{V} \leftarrow \textsc{SplitBlock}(\mathcal{V}, k)$
    $k \leftarrow k/2$
  end if
until F converges
output $\mathcal{S}$;

Algorithm 2: Parsimonious Black-box Attack

Baselines

- We evaluate the performance of our method comparing against **NES** and **Bandits**, which are the current state-of-the art in black-box attacks.
- We also show the white-box PGD results as the upper bound experiment.

Evaluation metric

- We quantify the performance in terms of **success rate**, **average queries**, and **median queries**.
- We further investigate the average queries on samples that NES, the weakest attack among the baselines, successfully fooled.

Cifar-10

Method	Success rate	Avg. queries	Med. queries	Avg. queries (NES success)
PGD (white-box)	47.2%	20	-	-
NES	29.5%	2872	900	2872
Bandits	38.6%	1877	459	520
Ours	**48.0%**	**1261**	**356**	**247**

Table: Results for ℓ_∞ untargeted attacks on Cifar-10. Maximum number of queries set to 20,000.

ImageNet

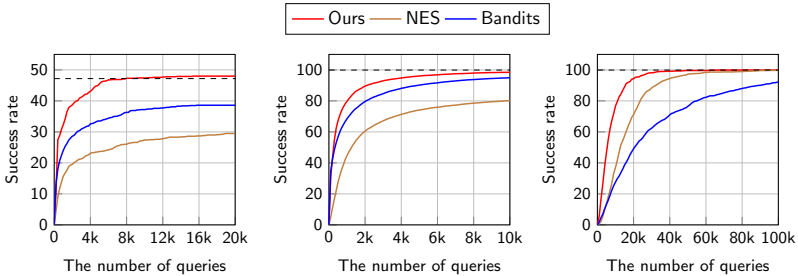
Method	Success rate	Avg. queries	Med. queries	Avg. queries (NES success)
PGD (white-box)	99.9%	20	-	-
NES†	77.8%	1735	-	1735
NES	80.3%	1660	900	1660
Bandits†	95.4%	1117	-	703
Bandits	94.9%	1030	286	603
Ours	**98.5%**	**722**	**237**	**376**

Table: Results for ℓ_∞ untargeted attacks on ImageNet. Maximum number of queries set to 10,000.

Method	Success rate	Avg. queries	Med. queries	Avg. queries (NES success)
PGD (white-box)	100%	200	-	-
NES†	99.2%	-	11550	-
NES	99.7%	16284	12650	16284
Bandits	92.3%	26421	18642	26421
Ours	**99.9%**	**7485**	**5373**	**7371**

Table: Results for ℓ_∞ targeted attacks on ImageNet. Maximum number of queries set to 100,000.

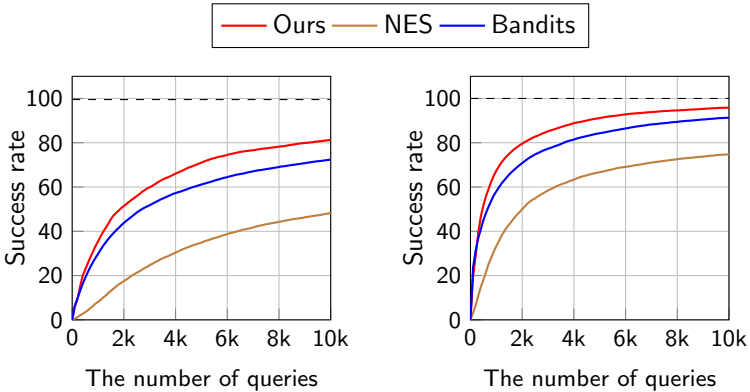
Queries vs Attack Success Rate plot



(a) Cifar-10, untargeted (b) ImageNet, untargeted (c) ImageNet, targeted

Figure: The cumulative distribution of the number of queries required for (a) untargeted attack on Cifar-10, (b) untargeted attack on ImageNet, and (c) targeted attack on ImageNet. The dashed line indicates the success rate of white-box PGD.

ImageNet with smaller ϵ



(a) $\epsilon = 0.01$ (b) $\epsilon = 0.03$

Figure: The cumulative distribution for the number of queries required for untargeted attack on ImageNet with (a) $\epsilon = 0.01$ and (b) $\epsilon = 0.03$.

Hyperparameter sensitivity

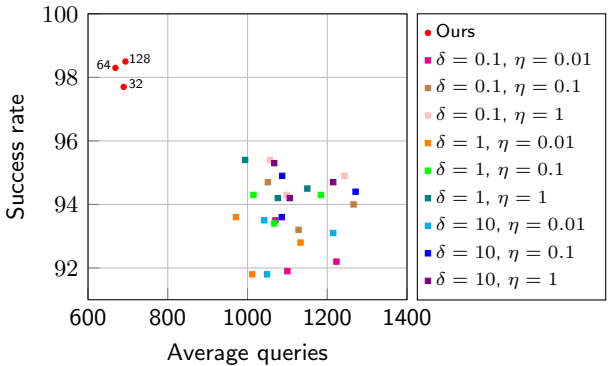


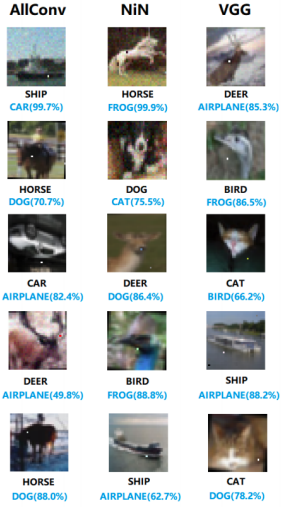
Figure: Success rate against the average number of queries with different hyperparameters. The square markers indicate the results of Bandit method. The numbers at round markers show the values of initial block size.

Conclusion

- We proposed an **efficient discrete surrogate** to the optimization problem which does not require estimating the gradient and consequently becomes free of the first order update hyperparameters to tune.
- Our experiments on Cifar-10 and ImageNet show the **state of the art black-box attack performance** with **significant reduction in the required queries** compared to a number of recently proposed methods.
- Github link: https://github.com/snu-mllab/parsimonious-blackbox-attack

Outline

One pixel attack for fooling deep neural networks



- One-pixel attacks that successfully fool deep neural networks trained on CIFAR-10 dataset.
- Attack success rate: Nin:72.85%, VGG:63.53%, AlexNet:41.23%

Su, Jiawei and Vargas, Danilo Vasconcellos and Kouichi, Sakurai, *One pixel attack for fooling deep neural networks, 2017*

One Pixel Attack

- Problem Description:

$$\underset{e(x)*}{\text{maximize}}\ f_{adv}(x + e(x)) \text{ subject to } ||e(x)||_0 \leqslant d$$

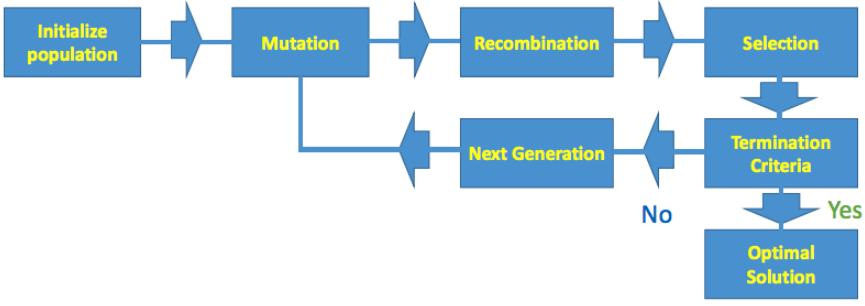
- In the case of one-pixel attack, d $= 1$.

One Pixel Attack

Differential evolution(DE):

- DE is a stochastic, population based optimization algorithm for solving optimization problems.
- DE belongs to the general class of evolutionary algorithm(EA).
- DE does not use the gradient information for optimizing (called black-box attack).

One Pixel Attack



- Consider a population size of N

$$x_{n,i}^{g} = [x_{n,1}^{g}, x_{n,2}^{g}, ..., x_{n,D}^{g}]$$

where g is the generation, and n =1,2,3,...,N

One Pixel Attack

- Define bounds for each variables
- Assume $x_{n,i}^{L}$, $x_{n,i}^{U}$ are lower and upper bound respectively.
- For example, in the case of cifar-10, D=5 and, the bounds are $[\underbrace{(0,31)}_{x},\underbrace{(0,31)}_{y},\underbrace{(0,255)}_{r},\underbrace{(0,255)}_{g},\underbrace{(0,255)}_{b}]$.

One Pixel Attack

- **Initial population**: generated randomly between upper lower and upper bound

$$x_{n,i} = x_{n,i}^{L} + rand() * (x_{n,i}^{U} - x_{n,i}^{L})$$

- **Mutation** : From each parameter vector, select three other vectors $x_{r1}^{g}, x_{r2}^{g}, x_{r3}^{g}$ randomly. Add the weighted difference of two of the vectors to the third.

$$v_{n}^{g+1} = x_{r1}^{g} + F * (x_{r2}^{g} - x_{r3}^{g})$$

F is generally taken between 0 and 1

One Pixel Attack

- **Recombination**:

$$u_{n,i}^{g+1} = \begin{cases} v_{n,i}^{g+1}, & \text{if } rand() \leqslant C_p \text{ or } i = I_{rand}. \\ x_{n,i}^{g}, & \text{if } rand() > C_p \text{ and } i \neq I_{rand}. \end{cases}$$

I_{rand} is a integer random number between [1,D]
C_p is the recombination probability

One Pixel Attack

- **Selection** :

$$x_n^{g+1} = \begin{cases} u_n^{g+1}, & \text{if } f(u_n^{g+1}) < f(x_n^g) \\ x_n^g, & \text{Otherwise} \end{cases}$$

- $[x^*, y^*, r^*, g^*, b^*] = \text{argmin}_{x_n^G} f(x_n^G)$