This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Trust Region Based Adversarial Attack on Neural Networks

Zhewei Yao[1] Amir Gholami[1] Peng Xu[2] Kurt Keutzer[1] Michael W. Mahoney[1]
[1]University of California, Berkeley [2] Stanford Univeristy
[1]{zheweiy, amirgh, keutzer, and mahoneymw}@berkeley.edu, [2]pengxu@stanford.edu

Abstract

Deep Neural Networks are quite vulnerable to adversarial perturbations. Current state-of-the-art adversarial attack methods typically require very time consuming hyper-parameter tuning, or require many iterations to solve an optimization based adversarial attack. To address this problem, we present a new family of trust region based adversarial attacks, with the goal of computing adversarial perturbations efficiently. We propose several attacks based on variants of the trust region optimization method. We test the proposed methods on Cifar-10 and ImageNet datasets using several different models including AlexNet, ResNet-50, VGG-16, and DenseNet-121 models. Our methods achieve comparable results with the Carlini-Wagner (CW) attack, but with significant speed up of up to 37×*, for the VGG-16 model on a Titan Xp GPU. For the case of ResNet-50 on ImageNet, we can bring down its classification accuracy to less than 0.1% with at most* 1.5% *relative* L_∞ *(or* L_2*) perturbation requiring only* 1.02 *seconds as compared to* 27.04 *seconds for the CW attack. We have open sourced our method which can be accessed at [1].*

1. Introduction

Deep Neural Networks (DNNs) have achieved impressive results in many research areas, such as classification, object detection, and natural language processing. However, recent studies have shown that DNNs are often not robust to adversarial perturbation of the input data [8, 26]. This has become a major challenge for DNN deployment, and significant research has been performed to address this. These efforts can be broadly classified into three categories: (i) research into finding strategies to defend against adversarial inputs (which has so far been largely unsuccessful); (ii) new attack methods that are stronger and can break the proposed defense mechanisms; and (iii) using attack methods as form of implicit adversarial regularization for training neural networks [23, 28, 29]. Our interest here is mainly focused on finding more effective attack methods that could be used in the latter two directions. Such ad-

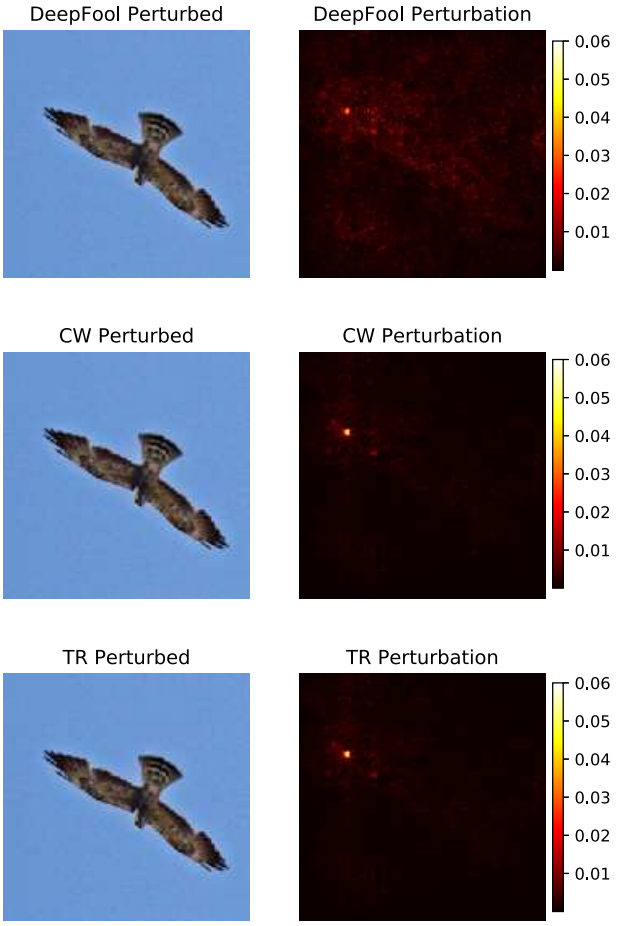

Figure 1: *An example of DeepFool, CW, and our TR attack on AlexNet, with* L_2 *norm. Both CW and TR perturbations are smaller in magnitude and more targeted than DeepFool's (*2× *smaller here). TR attack obtains similar perturbation as CW, but* **15**× *faster. In the case of the VGG-16 network, we achieve an even higher speedup of* **37.5**× *(please see Fig. 4 for timings).*

versarial attack methods can be broadly classified into two categories: white-box attacks, where the model architecture is known; and black-box attacks, where the adversary can only perform a finite number of queries and observe the model behaviour. In practice, white-box attacks are often not feasible, but recent work has shown that some adversarial attacks can actually transfer from one model to the other [15]. Therefore, precise knowledge of the target DNN may actually not be essential. Another important finding in this direction is the existence of an *adversarial patch*, i.e., a small set of pixels which, if added to an image, can fool the network. This has raised important security concerns for applications such as autonomous driving, where addition of such an adversarial patch to traffic signs could fool the system [3].

Relatedly, finding more *efficient* attack methods is important for evaluating defense strategies, and this is the main focus of our paper. For instance, the seminal work of [5] introduced a new type of optimization based attack, commonly referred to as CW (Carlini-Wagner) attack, which illustrated that defensive distillation [18] can be broken with its stronger attack. The latter approach had shown significant robustness to the fast gradient sign attack method [8], but not when tested against the stronger CW attack. Three metrics for an *efficient attack* are the speed with which such a perturbation can be computed, the magnitude or norm of the perturbation that needs to be added to the input to fool the network, and the transferability of the attack to other networks. Ideally (from the perspective of the attacker), a stronger attack with a smaller perturbation magnitude is desired, so that it could be undetectable (e.g. an adversarial patch that is harder to detect by humans).

In this work, we propose a novel trust region based attack method. Introduced in [6, 25], trust region (TR) methods form a family of numerical optimization methods for solving non-convex optimization problems [17]. The basic TR optimization method works by first defining a region, commonly referred to as *trust region*, around the current point in the optimization landscape, in which a (quadratic) model approximation is used to find a descent/ascent direction. The idea of using this confined region is due to the model approximation error. In particular, the trust region method is designed to improve upon vanilla first-order and second-order methods, especially in the presence of non-convexity.

We first consider a first-order TR method, which uses gradient information for attacking the target DNN model and adaptively adjusts the trust region. The main advantage of first-order attacks is their computational efficiency and ease of implementation. We show that our first-order TR method significantly reduces the over-estimation problem (i.e. requiring very large perturbation to fool the network), resulting in up to 3.9× reduction in the perturbation magnitude, as compared to DeepFool [16]. Furthermore, we show TR is significantly faster than the CW attack (up to 37×), while achieving similar attack performance. We then propose an adaptive TR method, where we adaptively choose the TR radius based on the model approximation to further speed up the attack process. Finally, we present the formulation for how our basic TR method could be extended to a second-order TR method, which could be useful for cases with significant non-linear decision boundaries, e.g., CNNs with Swish activation function [20]. In more detail, our main contributions are the following:

- We cast the adversarial attack problem into the optimization framework of TR methods. This enables several new attack schemes, which are easy to implement and are significantly more effective than existing attack methods (up to 3.9×, when compared to DeepFool). Our method requires a similar perturbation magnitude, as compared to CW, but it can compute the perturbation significantly faster (up to 37×), as it does not require extensive hyper-parameter tuning.
- Our TR-based attack methods can adaptively choose the perturbation magnitude in every iteration. This removes the need for expensive hyper-parameter tuning, which is a major issue with the existing optimization based methods.
- Our method can easily be extended to second-order TR attacks, which could be useful for non-linear activation functions. With fewer iterations, our second-order attack method outperforms the first-order attack method.

Limitations. We believe that it is important for every work to state its limitations (in general, but in particular in this area). We paid special attention to repeat the experiments multiple times, and we considered multiple different DNNs on different datasets to make sure the results are general. One important limitation of our approach is that a naïve implementation of our second-order method requires computation of Hessian matvec backpropogation, which is very expensive for DNNs. Although the second-order TR attack achieves better results, as compared to the first-order TR attack, this additional computational cost could limit its usefulness in certain applications. Moreover, our method achieves similar results as CW attack, but significantly faster. However, if we ignore the strength of the attack,

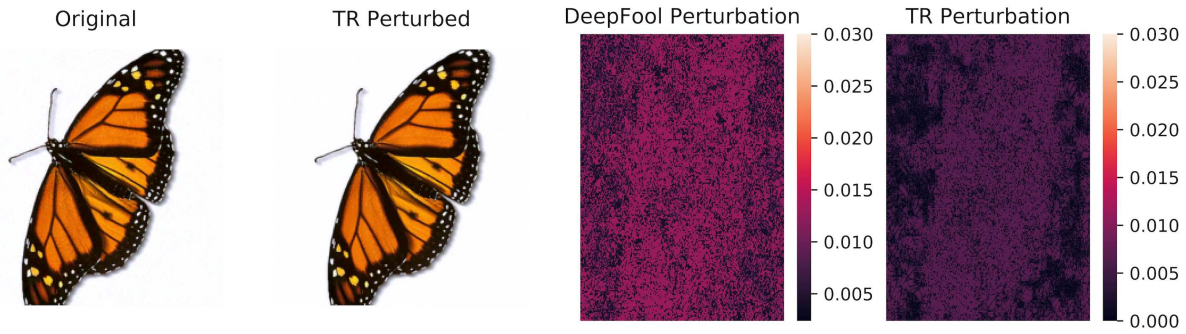

Figure 2: *An example of DeepFool and TR attack on VGG-16, with L_∞ norm. The first pattern is the original image. The second pattern is the image after TR attack. The final two patterns are the perturbations generated by DeepFool and TR. The TR perturbation is smaller than DeepFool's (1.9× smaller). Also, the TR perturbation is more concentrated around the butterfly.*

then the DeepFool attack is faster than our method (and CW's for that matter). Although such comparison may not be fair, as our attack is stronger. However, this may be an important point for certain applications where maximum speed is needed.

2. Background

In this section, we review related work on adversarial attacks. Consider $\mathbf{x} \in \mathbb{R}^n$ as input image, and $\mathbf{y} \in \mathbb{R}^c$ the corresponding label. Suppose $\mathcal{M}(\mathbf{x}; \theta) = \hat{\mathbf{y}}$ is the DNN's prediction probabilities, with θ the model parameters and $\hat{\mathbf{y}} \in \mathbb{R}^c$ the vector of probabilities. We denote the loss function of a DNN as $\mathcal{L}(\mathbf{x}, \theta, \mathbf{y})$. Then, an adversarial attack is aimed to find a (small) perturbation, $\Delta\mathbf{x}$, such that:

$$\arg\max(\mathcal{M}(\mathbf{x} + \Delta\mathbf{x}; \theta)) = \arg\max(\hat{\mathbf{y}}) \neq \arg\max(\mathbf{y}).$$

There is no closed form solution to analytically compute such a perturbation. However, several different approaches have been proposed by solving auxiliary optimization or analytical approximations to solve for the perturbation. For instance, the Fast Gradient Sign Method (FGSM) [8] is a simple adversarial attack scheme that works by directly maximizing the loss function $\mathcal{L}(\mathbf{x}, \theta, \mathbf{y})$. It was subsequently extended to an iterative FGSM [13], which performs multiple gradient ascend steps to compute the adversarial perturbation, and is often more effective than FGSM in attacking the network. Another interesting work in this direction is DeepFool, which uses an approximate analytical method. DeepFool assumes that the neural network behaves as an affine multiclass classifier, which allows one to find a closed form solution. DeepFool is based on "projecting" perturbed inputs to cross the decision boundary (schematically shown in Fig. 3), so that its classification is changed, and this was shown to outperform FGSM. However, the landscape of the decision boundary of a neural network is not linear. This is the case even for ReLU networks with the softmax layer. Even before the softmax layer, the landscape is *piecewise linear*, but this cannot be approximated with a simple affine transformation. Therefore, if we use the local information, we can overestimate/underestimate the adversarial perturbation needed to fool the network.

The seminal work of [5] introduced the so-called CW attack, a more sophisticated way to directly solve for the $\Delta\mathbf{x}$ perturbation. Here, the problem is cast as an optimization problem, where we seek to minimize the distance between the original image and the perturbed one, subject to the constraint that the perturbed input would be misclassified by the neural network. This work also clearly showed that defensive distillation, which at the time was believed to be a robust method to defend against adversaries, is not robust to stronger attacks. One major disadvantage of the CW attack is that it is very sensitive to hyper-parameter tuning. This is an important problem in applications where speed is important, as finding a good/optimal adversarial perturbation for a given input is very time consuming. Addressing this issue, without sacrificing attack strength, is a goal of our work.

On another direction, adversarial training has been used as a defense method against adversarial attacks [23]. In particular, by using adversarial examples during training, one can obtain models that are more robust to attacks (but still not foolproof). This adversarial training was further extended to ensemble adversarial training [27], with the goal of making the model more robust to black box attacks. Other approaches have also been proposed to detect/defend against adversarial attacks [14, 18]. However, it has recently been

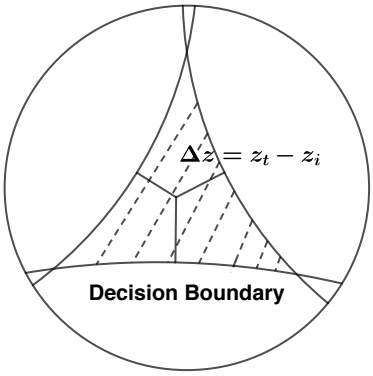

Figure 3: *Schematic illustration of the decision boundaries for a classification problem. Points mapped to the hashed region, are classified with the same label.*

shown that, with a stronger attack method, defense schemes such as distillation or obfuscated gradients can be broken [2, 4, 5].

A final important application of adversarial attacks is to train neural networks to obtain improved generalization, even in non-adversarial environments. Multiple recent works have shown that adversarial training (specifically, training with mixed adversarial and clean data) can be used to train a neural network from scratch in order to achieve a better final *generalization* performance [22, 23, 28, 29]. In particular, the work of [29] empirically showed that using adversarial training would result in finding areas with "flatter" curvature. This property has recently been argued to be an important parameter for generalization performance [11]. Here, the speed with which adversarial perturbations can be computed is very important since it appears in the inner loop of the training process, and the training needs to be performed for many epochs.

3. Trust Region Adversarial Attack

Let us denote the output of the DNN before the softmax function to be $\mathbf{z} = \mathbf{Z}(\mathbf{x};\theta) \in \mathbb{R}^c$. Therefore, we will have:

$$\mathcal{M}(\mathbf{x};\theta) = softmax\big(\mathbf{Z}(\mathbf{x};\theta)\big) = \hat{\mathbf{y}}.$$

Denote $\mathbf{y}_t$ to be the true label of $\mathbf{x}$, and $\mathbf{z}_t = \arg\max \mathbf{z}$ to be the prediction output of $\mathcal{M}(\mathbf{x};\theta)$. For clarification, note that $\mathbf{z}_t$ is only the same as $\mathbf{y}_t$ if the neural network makes the correct classification. An adversarial attack seeks to find $\Delta\mathbf{x}$ that fools the DNN, that is:

$$\arg\min_{\|\Delta\mathbf{x}\|_p} \arg\max \mathbf{Z}(\mathbf{x}+\Delta\mathbf{x};\theta) \neq \mathbf{y}_t, \qquad (1)$$

where $\|\cdot\|_p$ denotes the L_p norm of a vector. It is often computationally infeasible to solve (1) exactly.

Algorithm 1: Trust Region Attack

input : Image $\mathbf{x}^0$, label $\mathbf{y}$, initial radius ϵ^0, threshold σ_1 and σ_2, radius adjustment rate η
output : Adversarial Image

$\Delta\mathbf{x} = 0$, $j = 0$; // Initialization
Using scheme to choose the attacking index i;
// Index Selection
while $\arg\max \mathbf{Z}(\mathbf{x}^j) = \arg\max \mathbf{y}$ **do**
  $\Delta\mathbf{x}_{tmp} = \arg\min_{\|\Delta\mathbf{x}^j\| \le \epsilon^j} m^j(\Delta\mathbf{x}^j) =$
  $\arg\min_{\|\Delta\mathbf{x}^j\| \le \epsilon^j} \langle \Delta\mathbf{x}^j, \mathbf{g}^j_{t,i}\rangle + \frac{1}{2}\langle \Delta\mathbf{x}^j, \mathbf{H}^j_{t,i}\Delta\mathbf{x}^j\rangle$
  $\mathbf{x}^{j+1} = clip(\mathbf{x}^j + \Delta\mathbf{x}_{tmp}, \min(\mathbf{x}), \max(\mathbf{x}))$;
  // Update
  $\rho = \frac{(\mathbf{z}_t^{j+1} - \mathbf{z}_i^{j+1}) - (\mathbf{z}_t^{j+1} - \mathbf{z}_i^{j+1})}{m^j(\Delta\mathbf{x}^j)}$; // Compute Ratio
  if $\rho > \sigma_1$ **then**
    $\epsilon^{j+1} = \min\{\eta\epsilon^j, \epsilon_{\max}\}$
  else if $\rho < \sigma_2$ **then**
    $\epsilon^{j+1} = \min\{\epsilon^j/\eta, \epsilon_{\min}\}$
  else
    $\epsilon^{j+1} = \epsilon^j$
  $j = j + 1$

Therefore, a common approach is to approximately solve (1) [5, 8, 26]. To do so, the problem can be formulated as follows:

$$\max_{\|\Delta\mathbf{x}\|_p \le \epsilon} \mathcal{J}(\mathbf{x}+\Delta\mathbf{x}, \theta, \mathbf{y}), \qquad (2)$$

where ϵ constrains the perturbation magnitude, and $\mathcal{J}$ can be either the loss function ($\mathcal{L}$) or more generally another kernel [5]. In the case of DeepFool (DF) attack, this problem is solved by approximating the decision boundary by a linear affine transformation. For such a decision boundary, the perturbation magnitude could be analytically computed by just evaluating the gradient at the current point. However, for neural networks this approximation could be very inaccurate, that is it could lead to over/under-estimation of the perturbation along a sub-optimal direction. The smallest direction would be orthogonal to the decision boundary, and this cannot be computed by a simple affine transformation, as the decision boundary is non-linear (see Fig. 3 for illustration). This is obvious for non-linear activation functions, but even in the case of ReLU, the model behaves like a piece-wise linear function (before the softmax layer, and actually non-linear afterwards). This approach cannot correctly find the orthogonal direction, even if we ignore the non-linearity of the softmax layer.

To address this limitation, we instead use TR methods, which are well-known for solving non-convex optimization problems [25]. The problem of finding adver-

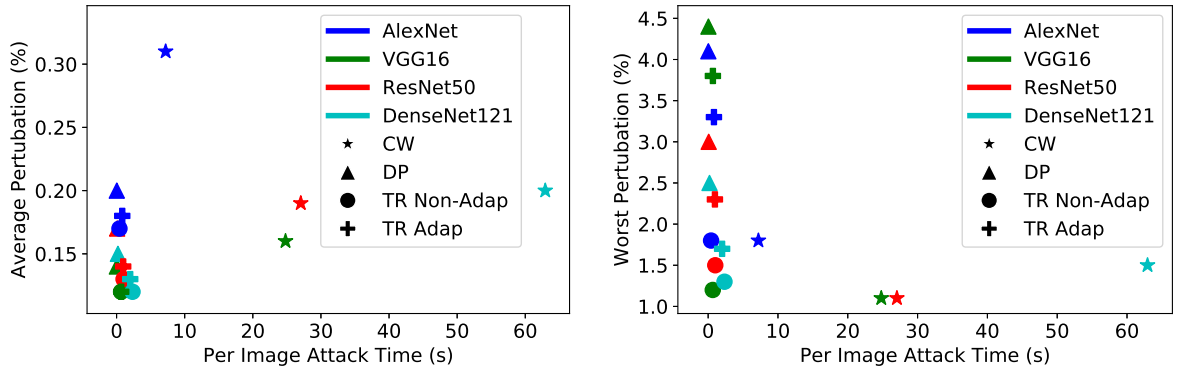

Figure 4: *The two subfigures show, for various neural networks, the time to compute the adversarial attack (x-axis) and the perturbation needed by that attack method to fool an image (y-axis), corresponding to ImageNet results in Table 3. On the left, the y-axis is plotted for average perturbation; and on the right, for the worst case perturbation. An attack that achieves smaller perturbation in shorter time is preferred. Different colors represent different models, and different markers illustrate the different attack methods. Observe that our TR and TR Adap methods achieve similar perturbations as CW but with significantly less time (up to* $\mathbf{37.5}\times$*).*

sarial perturbation using TR is defined as follows:

$$\max_{\|\Delta \mathbf{x}^j\|_p \leq \epsilon^j} m^j(\Delta \mathbf{x}^j) = \langle \Delta \mathbf{x}^j, \mathbf{g}_{t,i}^j \rangle + \frac{1}{2} \langle \Delta \mathbf{x}^j, \mathbf{H}_{t,i}^j \Delta \mathbf{x}^j \rangle, \tag{3}$$

where ϵ^j is the TR radius at j^{th} iteration, m^j is the approximation of the kernel function of $f(\mathbf{x}^{j-1}) = (\mathbf{z}_t^{j-1} - \mathbf{z}_i^{j-1})$ with $\mathbf{g}_{t,i}^j$ and $\mathbf{H}_{t,i}^j$ denoting the corresponding gradient and Hessian, and $\mathbf{x}^j = \mathbf{x} + \sum_{i=1}^{j-1} \Delta \mathbf{x}^i$. Note that the subscript means the index of maximal $\mathbf{z}$, i.e. $\arg\max \mathbf{z}$. The main idea of the TR method is to iteratively select the trusted radius ϵ^j to find the adversarial perturbation within this region such that the probability of an incorrect class becomes maximum. TR adjusts this radius by measuring the approximation of the local model $m^j(\mathbf{s}^j)$ to the actual function value $f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)$. In particular, we increase the trusted radius if the approximation of the function is accurate (measured by $\rho = \frac{f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)}{m^j(\mathbf{s}^j)} > \sigma_1$ with a typical value of $\sigma_1 = 0.9$). In such a case, the trusted radius is increased for the next iterations by a factor of $\eta > 1$ ($\epsilon^{j+1} = \eta \epsilon^t$). However, when the local model $m^j(\mathbf{s}^j)$ is a poor approximation of $f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)$, i.e., $\rho = \frac{f(\mathbf{x}^{j+1}) - f(\mathbf{x}^j)}{m^j(\mathbf{s}^j)} < \sigma_2$ (with a typical $\sigma_2 = 0.5$), we decrease the trusted radius for the next iteration $\epsilon^{j+1} = \epsilon^t / \eta$. Otherwise, we keep the same ϵ^j for ϵ^{j+1}. Typically, a threshold is also used for lower and upper bounds of ϵ^j. Using this approach, the TR attack can iteratively find an adversarial perturbation to fool the network. See Alg. 1 for details.

Note that for cases where all the activations of the DNN are ReLU, the Hessian is zero almost everywhere [29, Theorem 1], and we actually do not need the Hessian. This means the landscape of $\mathbf{z}_t - \mathbf{z}_i$ is piece-wise linear, i.e., we could omit $\mathbf{H}_{t,i}^j$ in (3). However, for non-linear activation functions, we need to keep the Hessian term (since when the NN has smooth activation functions, the Hessian is not zero). For these cases, the problem of finding the adversarial perturbation becomes a Quadratic Constrained Quadratic Programming (QCQP) problem. It is quadratic constraint due to the fact that the norm of the perturbation is limited by the TR radius, η^j, and the quadratic programming arises from the non-zero Hessian term. We use Lanczos algorithm to solve the QCQP problem. In this approach, the solution is iteratively found in a Krylov subspace formed by the Hessian operator.

4. Performance of the Method

To test the efficacy of the TR attack method and to compare its performance with other approaches, we perform multiple experiments using different models on Cifar-10 [12] and ImageNet [7] datasets. In particular, we compare to DeepFool [16], iterative FGSM [8, 13], and the Carlini-Wagner (CW) attack [5].

As mentioned above, the original TR method adaptively selects the perturbation magnitude. Here, to test how effective the adaptive method performs, we also experiment with a case where we set the TR radius to be a fixed small value and compare the results with the original adaptive version. We refer to the fixed radius version as "TR Non-Adap" and the adaptive version as "TR Adap". Furthermore, the metric that we use for performance of the attack is the relative perturbation,

Table 1: *Average perturbations / worst case perturbations are reported of different models on Cifar-10 for best class attack.. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.*

		DeepFool	CW	TR Non-Adap	TR Adap
Model	Accuracy	ρ_2	ρ_2	ρ_2	ρ_2
AlexLike	85.78	1.67% / 11.5%	1.47% / 9.70%	1.49% / 9.13%	1.49% / 9.09%
AlexLike-S	86.53	1.74% / 11.0%	1.57% / 8.59%	1.57% / 9.48%	1.57% / 9.46%
ResNet	92.10	0.80% / 5.60%	0.62% / 3.12%	0.66% / 3.97%	0.66% / 3.96%
WResNet	94.77	0.89% / 5.79%	0.66% / 4.51%	0.73% / 4.43%	0.72% / 4.34%
		DeepFool	FGSM	TR Non-Adap	TR Adap
Model	Accuracy	ρ_∞	ρ_∞	ρ_∞	ρ_∞
AlexLike	85.78	1.15% / 6.85%	1.40% / 16.44%	1.05% / 5.45%	1.03% / 5.45%
AlexLike-S	86.53	1.18% / 6.01%	1.45% / 14.88%	1.09% / 4.76%	1.07% / 4.73%
ResNet	92.10	0.60% / 3.98%	0.85% / 4.35%	0.56% / 3.18%	0.50% / 3.35%
WResNet	94.77	0.66% / 3.34%	0.85% / 3.30%	0.56% / 2.67%	0.54% / 2.69%

defined as follows:

$$\rho_p = \frac{\|\Delta \mathbf{x}\|_p}{\|\mathbf{x}\|_p}, \tag{4}$$

where $\Delta \mathbf{x}$ is the perturbation needed to fool the testing example. The perturbation is chosen such that the accuracy of the model is reduced to less than 0.1%. We report both the average perturbation as well as the highest perturbation required to fool a testing image. To clarify this, the highest perturbation is computed after all of testing images (50K in ImageNet and 10K in Cifar-10) and then finding the the highest perturbation magnitude that was needed to fool a correctly classified example. We refer to this case as *worst case* perturbation. Ideally we would like this worst case perturbation to be bounded and close to the average cases.

Table 2: *Average perturbations / worst case perturbations are reported of different models on Cifar-10 for hardest class attack. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.*

	DeepFool	TR Non-Adap	TR Adap
Model	ρ_2	ρ_2	ρ_2
AlexLike	4.36% /18.9%	2.47% /13.4%	2.47% /13.4%
AlexLike-S	4.70% /17.7%	2.63% /14.4%	2.62% /14.2%
ResNet	1.71% /8.01%	0.99% /4.76%	0.99% /4.90%
WResNet	1.80% /8.74%	1.05% /6.23%	1.08% /6.23%
Model	ρ_∞	ρ_∞	ρ_∞
AlexLike	2.96% /12.6%	1.92% /9.99%	1.86% /10.0%
AlexLike-S	3.12% /12.2%	1.98% /8.19%	1.92% /8.17%
ResNet	1.34% /9.65%	0.77% /4.70%	0.85% /5.44%
WResNet	1.35% /6.49%	0.81% /3.77%	0.89% /3.90%

4.1. Setup

We consider multiple different neural networks including variants of (wide) residual networks [9, 30], AlexNet, VGG16 [24], and DenseNet from [10]. We also test with custom smaller/shallower convolutional networks such as a simple CNN [29, C1] (refer as AlexLike with ReLU and AlexLike-S with Swish activation). To test the second order attack method we run experiments with AlexNet-S (by replacing all ReLUs with Swish function activation function [19]), along with a simple MLP ($3072 \to 1024 \to 512 \to 512 \to 256 \to 10$) with Swish activation function.

By definition, an adversarial attack is considered successful if it is able to change the classification of the input image. Here we perform two types of attacks. The first one is where we compute the smallest perturbation needed to change the target label. We refer to this as *best class* attack. This means we attack the class with:

$$\arg\min_j \frac{z_t - z_j}{\|\nabla_{\mathbf{x}}(z_t - z_j)\|}.$$

Intuitively, this corresponds to perturbing the input to cross the closest decision boundary (Fig. 3). On the other hand, we also consider perturbing the input to the class whose decision boundary is farthest away:

$$\arg\max_j \frac{z_t - z_j}{\|\nabla_{\mathbf{x}}(z_t - z_j)\|}.$$

Furthermore, we report two perturbation metrics of *average perturbation*, computed as:

$$\rho_p = \frac{1}{N}\sum_{i=1}^{N} \frac{\|\Delta \mathbf{x}_i\|_p}{\|\mathbf{x}_i\|_p},$$

Table 3: *Average perturbations / worst case perturbations are reported of different models on ImageNet for best class attack. Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.*

		DeepFool	CW	TR Non-Adap	TR Adap
Model	Accuracy	ρ_2	ρ_2	ρ_2	ρ_2
AlexNet	56.5	0.20% / 4.1%	0.31% / 1.8%	0.17% / 2.5%	0.18% / 3.3%
VGG16	71.6	0.14% / 4.4%	0.16% / 1.1%	0.12% / 1.2%	0.12% / 3.8%
ResNet50	76.1	0.17% / 3.0%	0.19% / 1.1%	0.13% / 1.5%	0.14% / 2.3%
DenseNet121	74.4	0.15% / 2.5%	0.20% / 1.5%	0.12% / 1.3%	0.13% / 1.7%
		DeepFool	FGSM	TR Non-Adap	TR Adap
Model	Accuracy	ρ_∞	ρ_∞	ρ_∞	ρ_∞
AlexNet	56.5	0.14% / 4.3%	0.16% / 4.7%	0.13% / 1.4%	0.13% / 3.6%
VGG16	71.5	0.11% / 4.0%	0.18% / 5.1%	0.10% / 1.4%	0.10% / 3.4%
ResNet50	76.1	0.13% / 3.2%	0.18% / 3.7%	0.11% / 1.3%	0.11% / 2.7%
DenseNet121	74.4	0.11% / 2.3%	0.15% / 4.1%	0.10% / 1.1%	0.10% / 1.8%

along with worst perturbation, computed as:

$$\rho_p = \max\{\frac{\|\Delta \mathbf{x}_i\|_p}{\|\mathbf{x}_i\|_p}\}_{i=1}^N.$$

For comparison, we also consider the following attack methods:

- Iterative FGSM from [8, 13], where the following formula is used to compute adversarial perturbation, after which the perturbation is clipped in range $(\min(\mathbf{x}), \max(x))$:

$$\mathbf{x}^{j+1} = \mathbf{x}^j + \epsilon \; sign(\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}^j, \theta, \mathbf{y})),$$

- DeepFool (DF) from [16]. We follow the same implementation as [16]. For the hardest class test, the target class is set the same as our TR method.

- CW attack from [5]. We use the open source code from [21][1].

Finally, we measure the time to fool an input image by averaging the attack time over all the testing examples. The measurements are performed on a Titan Xp GPU with an Intel E5-2640 CPU.

4.2. Cifar-10

We first compare different attacks of various neural network models on Cifar-10 dataset, as reported in Table 1. Here, we compute the average and worst case perturbation for best class attack. For L_2 attack, we can see that TR Non-Adap can achieve comparable perturbation as CW, with both TR and CW requiring smaller perturbation than DeepFool. An important advantage of the TR attack is its speed, as compared to CW attack, which is illustrated in Fig. **??** (please see appendix). Here we plot the time spent to fool one input image versus average perturbation for all L_2 attack methods on different models. It can be clearly seen that, with similar perturbations, the time to get the adversarial examples is: TR $<$ CW. Note that DeepFool is also very fast but requires much larger perturbations than TR attack and CW. Also note that the TR Adap method achieves similar results, with slightly slower speed and slightly larger perturbation. This is because the adaptive method has not been tuned any way, whereas for the non adaptive version we manually tuned ϵ. TR Adap does not require tuning, as it automatically adjust the TR radius. The slight performance degradation is due to the relaxed σ_1 and σ_2 parameters, which could

Table 4: *Average perturbations / worst case perturbations are reported of different models on ImageNet for hardest class attack (on the top 100 prediction classes). Lower values are better. The first set of rows show L_2 attack and the second shows L_∞ attack.*

	DeepFool	TR Non-Adap	TR Adap
Model	ρ_2	ρ_2	ρ_2
AlexNet	0.74% /8.7%	0.39% /5.0%	0.39% /5.0%
VGG16	0.45% /5.4%	0.27% /3.6%	0.27% /3.8%
ResNet50	0.52% /5.8%	0.31% /4.2%	0.31% /4.2%
DenseNet	0.48% /5.7%	0.29% /3.8%	0.29% /3.8%
Model	ρ_∞	ρ_∞	ρ_∞
AlexNet	0.53% /9.9%	0.31% /7.5%	0.33% /9.1%
VGG16	0.36% /11.6%	0.25% /5.1%	0.26% /6.8%
ResNet50	0.43% /6.6%	0.28% /3.7%	0.30% /4.6%
DenseNet	0.38% /6.4%	0.24% /4.5%	0.27% /5.7%

[1] https://github.com/bethgelab/foolbox

Table 5: *Second order and first order comparison on MLP and AlexNet with Swish activation function on Cifar-10. The corresponding baseline accuracy without adversarial perturbation is 62.4% and 76.6%, respectively. As expected, the second order TR attack achieves better results as compared to first-order with fixed iterations. However, the second-order attack is significantly more expensive, due to the overhead of solving QCQP problem.*

Iter		1	2	3	4	5	6	7	8	9	10
MLP	TR First	47.63	33.7	22.24	13.76	8.13	4.59	2.41	1.31	0.63	0.27
MLP	TR Second	47.84	33.37	21.49	13.3	7.39	4.16	2.17	1.09	0.49	0.20
AlexNet	TR First	51.51	28.17	12.45	5.53	2.61	1.33	0.82	0.66	0.51	0.46
AlexNet	TR Second	50.96	26.97	10.73	4.11	1.79	0.91	0.67	0.54	0.47	0.44

be made more conservative as a trade-off for speed. But we did not tune these parameters beyond the default, to give a realistic performance for the non-tuned version.

Another important test is to measure the perturbation needed to fool the network to the hardest target class. This is important in that flipping a pedestrian to a cyclist may be easier than flipping it to become a car. In Table 2, we report the hardest class attack on Cifar-10. Note that our methods are roughly 1.5 times better than DeepFool in all cases. Particularly, For L_2 attack on WResNet, our worst case is 3.9 times better than DeepFool in terms of perturbation magnitude.

4.3. ImageNet Result

We observe similar trends on ImageNet. We report different attacks on various models on ImageNet in Table 3. Note that TR and CW require significantly smaller perturbation for the worst case as compared to DeepFool. However, TR is significantly faster than CW. The timing results are shown in Fig. 4. For instance in the case of VGG-16, TR attack is **37.5**× faster than CW which is significant. An example perturbation with AlexNet is shown in Fig. 1 (for which TR is **15**× faster). As one can see, CW and TR perturbations are smaller than DeepFool (2× in this case), and more targeted around the object. For L_∞ methods, our TR Non-Adap and TR Adap are consistently better than FGSM and DeepFool in both average and worst cases. Particularly, for worst cases, TR is roughly two times better than the other methods. An example perturbation of DeepFool and TR Non-Adap with L_∞ on VGG16 is shown in Fig. 2. It can be clearly seen that, TR perturbation is much smaller than DeepFool (1.9× in this case), and more targeted around the objective.

4.4. Second order method

As mentioned in Section 3, the ReLU activation function does not require Hessian computation. However, for non-linear activation functions including Hessian information is beneficial, although it may be very expensive. To test this hypothesis, we consider two models with Swish activation function. We fix the TR radius (set to be 1 for all cases) of our first and second order methods, and gradually increase the iterations. Table 5 shows the results for MLP and AlexNet models. It can be seen that second order TR out-performs the first order TR method in all iterations. Particularly, for two and three iterations on AlexNet, TRS can drop the model accuracy 1.2% more as compared to the first order TR variant. However, the second order based model is more expensive than the first order model, mainly due to the overhead associated with solving the QCQP problem. There is no closed form solution for this problem because the problem is non-convex and the Hessian can contain negative spectrum. Developing a computationally efficient method for this is an interesting next direction.

5. Conclusions

We have considered various TR based methods for adversarial attacks on neural networks. We presented the formulation for the TR method along with results for our first/second-order attacks. We considered multiple models on Cifar-10 and ImageNet datasets, including variants of residual and densely connected networks. Our method requires significantly smaller perturbation (up to 3.9×), as compared to DeepFool. Furthermore, we achieve similar results (in terms of average/worst perturbation magnitude to fool the network), as compared to the CW attack, but with significant speed up of up to 37.5×. For all the models considered, our attack method can bring down the model accuracy to less than 0.1% with relative small perturbation (in L_2/L_∞ norms) of the input image. Meanwhile, we also tested the second order TR attack by backpropogating the Hessian information through the neural network, showing that it can find a stronger attack direction, as compared to the first order variant.

References

[1] https://github.com/amirgholami/trattack, November 2018.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[3] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[4] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.

[5] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[6] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*, volume 1. Siam, 2000.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[8] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (arXiv:1412.6572)*, 2015.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[10] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, volume 1, page 3, 2017.

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[12] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[13] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[14] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.

[15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94. IEEE, 2017.

[16] Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, number EPFL-CONF-218057, 2016.

[17] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 2006.

[18] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[19] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Swish: a self-gated activation function. *arXiv preprint arXiv:1710.05941*, 2017.

[20] Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2018.

[21] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox v0. 8.0: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*, 2017.

[22] Swami Sankaranarayanan, Arpit Jain, Rama Chellappa, and Ser Nam Lim. Regularizing deep networks using efficient layerwise adversarial training. *arXiv preprint arXiv:1705.07819*, 2017.

[23] Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 2018.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[25] Trond Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[27] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[28] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. Large batch size training of neural networks with adversarial training and second-order information. *arXiv preprint arXiv:1810.01021*, 2018.

[29] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. *Neural Information Processing Systems (NIPS'18)*, 2018.

[30] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.