Adversarial Examples: Attacks and Defenses for Deep Learning

Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, Xiaolin Li*
National Science Foundation Center for Big Learning, University of Florida
{chbrian, pan.he, valder, rbhat}@ufl.edu, andyli@ece.ufl.edu

***Abstract*—With rapid progress and great successes in a wide spectrum of applications, deep learning is being applied in many safety-critical environments. However, deep neural networks have been recently found vulnerable to well-designed input samples, called *adversarial examples*. Adversarial examples are imperceptible to human but can easily fool deep neural networks in the testing/deploying stage. The vulnerability to adversarial examples becomes one of the major risks for applying deep neural networks in safety-critical scenarios. Therefore, the attacks and defenses on adversarial examples draw great attention. In this paper, we review recent findings on adversarial examples against deep neural networks, summarize the methods for generating adversarial examples, and propose a taxonomy of these methods. Under the taxonomy, applications for adversarial examples are investigated. We further elaborate on countermeasures for adversarial examples and explore the challenges and the potential solutions.**

***Index Terms*—deep neural network, deep learning, security, adversarial examples**

I. Introduction

Deep neural networks (deep learning) have made significant progresses in a wide domain of machine learning: image classification (object recognition) [1], [2], object detection [3], [4], speech recognition [5], language translation [6], voice synthesis [7]. The online Go Master (AlphaGo [8]) beat more than 50 top go players in the world. Recently AlphaGo Zero [9] surpassed its previous version without using human knowledge and a general version, AlphaZero [10], achieved a superhuman level within 24 hours of cross domains of chess, Shogi, and Go.

Deep learning requires less hand engineered features and expert knowledge. Driven by the emergence of big data and hardware acceleration, the intricacy of data can be extracted with higher and more abstract level representation from raw input features [11].

Constantly increasing number of real-world applications and systems have been powered by deep learning. Companies from IT to the auto industry (*e.g.*, Google, Telsa, Mercedes, and Uber) are testing self-driving cars, which require plenty of deep learning techniques such as object detection, reinforcement learning, and multimodal deep learning. Face recognition system has been deployed in ATMs as a method of biometric authentication in China [12]. Apple provides face authentication to unlock mobile phones [13]. Behavior-based malware detection and anomaly detection solutions are built upon deep learning to find semantic features [14]–[17].

Despite great successes in numerous applications, many of these deep learning empowered applications are life crucial, raising great concerns in the field of safety and security. "With great power comes great responsibility" [18]. Recent studies find that deep learning is vulnerable against well-designed input samples. These samples can easily fool a well-performed deep learning model with little perturbations imperceptible to humans.

Szegedy *et al.* first generated small perturbations on the images for image classification problem and fooled state-of-the-art deep neural networks with high probability [19]. These misclassified samples were named *Adversarial Examples*.

A large amount of deep learning based applications have been used or planned to be deployed in the physical world, especially in the safety-critical environments. Recent studies show that adversarial examples can be applied to real world. Thus, adversarial examples require to be addressed carefully. For instance, an adversary can construct physical adversarial examples and confuse the autonomous vehicles by manipulating a stop sign in a traffic sign recognition system [20], [21] or removing the segmentation of pedestrians in an object detection system [22]. Adversarial commands can be generated by attackers against automatic speech recognition (ASR) models and Voice Controllable System (VCS) [23], [24] such as Apple Siri [25], Amazon Alexa [26], and Microsoft Cortana [27].

Deep learning is widely regarded as a black box — we all know that it performs well, but with limited knowledge of the reason [28], [29]. Many studies have been proposed to explain and interpret deep neural networks [30]–[33]. From inspecting adversarial examples, we may gain insights on semantic inner levels of neural networks [34] and find problematic decision boundaries, which in turn helps to increase robustness and performance of neural networks [35] and improve the interpretability [36].

In this paper, we investigate and summarize the approaches for generating adversarial examples, applications for adversarial examples and the corresponding countermeasures. We explore the characteristics and the possible causes of adversarial examples. Recent advances in deep learning revolve around supervised learning, especially in the field of computer vision task. Therefore, most of adversarial examples are generated against computer vision models. We mainly discuss the adversarial examples for the image classification/object recognition

* Corresponding author.

task in this paper. Adversarial examples for other tasks will be investigated in Section V.

Inspired by [37], we define the **Threat Model** in this paper as follows:

- The adversaries can attack only at the testing/deploying stage. They can tamper only the input data in the testing stage after the victim deep learning model is trained. Neither the trained model or the training dataset can be modified. The adversaries may have knowledge of the trained model (architectures and parameters) but not allowed to modify the model, which is a common assumption for many online machine learning services. Attacking at the training stage (*e.g.*, training data poisoning) is another interesting topic and has been studied in [38]–[43]. Due to the limitation of space, we do not include this topic in the paper.
- Since the great performance achieved by deep learning, we only study the attacks against deep neural network-based models. Adversarial examples against conventional machine learning have been discussed in previous studies (Section II). Adversarial examples against deep neural networks are proved effective to conventional machine learning models [44] (see Section VII-A).
- The adversaries target only *integrity*. Integrity is presented by the performance metrics (*e.g.*, accuracy, F1 score, AUC), which is essential to a deep learning model. Although other security issues related to confidentiality and privacy have been drawn attention in deep learning [45]–[47], we focus on the attacks that degrade the performance of deep learning models: attacks cause the increasing number of false positives and false negatives.

The notations and symbols used in this paper are listed in Table I.

Table I: Notation and symbols used in this paper

Notations and Symbols	Description
x	original (clean, unmodified) input data
l	label of class in the classification problem. $l = 1, 2, \ldots, m$, where m is the number of classes
x'	adversarial example (modified input data)
l'	label of the adversarial class in targeted adversarial examples
$f(\cdot)$	deep learning model (for the image classification task, $f \in F : \mathbb{R}^n \rightarrow l$)
θ	parameters of deep learning model f
$J(\cdot)$	loss function (*e.g.*, cross-entropy)
η	difference between original and modified input data: $\eta = x' - x$ (the exact same size as the input data)
$\|\cdot\|_p$	ℓ_p norm
∇	gradient
$H(\cdot)$	Hessian, the second-order of derivatives
$KL(\cdot)$	Kullback-Leibler (KL) divergence function

An instance of adversarial examples for image classification task: Using a trained image classifier published by a third party, a user inputs one image to get the prediction of class label. Adversarial images are original clean images with small perturbations, often barely recognizable by humans. However, such perturbations misguide the image classifier. The user will get a response of an incorrect image label.

Given a trained deep learning model f, an original input data sample x, generating an adversarial example x' can generally be described as a box-constrained optimization problem:

$$\begin{aligned} \min_{x'} \quad & \|x' - x\| \\ s.t. \quad & f(x') = l', \\ & f(x) = l, \\ & l \neq l', \\ & x' \in [0, 1], \end{aligned} \tag{1}$$

where l and l' denote the output label of x and x', $\|\cdot\|$ denotes the distance between two data sample. Let $\eta = x' - x$ be the perturbation added on x. This optimization problem minimizes the perturbation while misclassifying the prediction with a constraint of input data. Many variants of this optimization problem have been proposed in different scenarios and assumptions (see Section IV). For instance, some adversaries consider that if $\|\eta\| \leq \delta$, the perturbation is small enough to be unnoticeable to human. Therefore, the perturbation is viewed as a constraint. The optimization objective function becomes the distance of targeted prediction score from the original prediction score. This variant problem can be described by:

$$\begin{aligned} \min_{x'} \quad & J(f(x'), l') \\ s.t. \quad & \|\eta\| \leq \epsilon, \\ & f(x) = l, \\ & l \neq l', \end{aligned} \tag{2}$$

where ϵ denotes the measurement of the perturbation constraint.

This paper presents the following contributions:

- To systematically analyze the approaches for generating adversarial examples, we taxonomize attack approaches along different axes to provide an accessible and intuitive overview of these approaches.
- We investigate recent approaches and their variants for generating adversarial examples and compare them using the proposed taxonomy. We show examples of selected applications from the field of reinforcement learning, generative modeling, face recognition, object detection, semantic segmentation, natural language processing, and malware detection. The countermeasures for adversarial examples are also discussed.
- We outline main challenges and potential future research directions for adversarial examples, around three main problems: the transferability of adversarial examples, the existence of adversarial examples, and the robustness evaluation of deep neural networks.

The remaining of this paper is organized as follows. Section II introduces the background of deep learning techniques, models, and datasets. We discuss adversarial examples raised in conventional machine learning in Section II. We propose a taxonomy of approaches for generating adversarial examples in Section III and elaborate on these approaches in Section IV. In Section V, we discuss the applications for adversarial examples. Corresponding countermeasures are investigated in Section VI. We discuss the current challenges and potential solutions in Section VII. Section VIII concludes the work.

II. Background

In this section, we briefly introduce deep learning techniques, most of which, to some extent, are used in generating adversarial examples. Next, we review adversarial examples in the era of conventional ML and compared the difference between adversarial examples in conventional ML and that in DL.

A. Brief Introduction to Deep Learning

This subsection discusses main concepts, new techniques, popular architectures, and common datasets in deep learning. Due to the wide use and breakthrough successes, these findings become the major targets of attacks, where adversaries are usually applied to evaluate the attack methods.

1) Main concepts in deep learning: Deep learning is a type of machine learning that makes computers to learn from experience and knowledge without explicit programming and extract useful patterns from raw data. For conventional machine learning algorithms, however, it is difficult to extract well-represented features due to the limitations, such as curse of dimensionality [48], computational bottleneck [49], and requirement of domain and expert knowledge. Deep neural network is a machine learning algorithm with many layers ("deep") of networks. Deep learning solves the problem of representation by building multiple simple features to represent a sophisticated concept. For example, a deep learning-based image classification system represents an object by describing edges, fabrics, and structures in the hidden layers. As the increasing number of available training data, deep learning becomes more powerful. Deep learning models solve complicated problems by complex and large models, with the help of hardware acceleration in computational time.

A neural network layer is composed of a set of perceptrons (artificial neurons). Each perceptron maps a set of inputs to output values with a simple activation function. The function of a neural network is formed in a chain:

$$f(x) = f^{(k)}(\cdots f^{(2)}(f^{(1)}(x))), \tag{3}$$

where $f^{(i)}$ is the function of the ith layer of the network, $i = 1, 2, \cdots k$.

Convolutional neural networks (CNNs) and **Recurrent neural networks (RNNs)** are two most widely used neural networks in recent neural network architectures. CNNs deploy convolution operations on hidden layers for weight sharing and parameter reduction. CNNs can extract local information from grid-like input data. CNNs have shown incredible successes in computer vision tasks, such as image classification [1], [50], object detection [4], [51], and semantic segmentation [52], [53]. RNNs are neural networks for processing sequential input data with variable length. RNNs produce an output at each time step. The hidden neuron at each time step is calculated based on input data and hidden neurons at previous time step. To avoid vanishing/exploding gradients of RNNs in long-term dependency, long short-term memory (LSTM) and Gated Recurrent Unit (GRU) with controllable gates are widely used in practical applications.

Generative Adversarial Network (GAN) is a type of generative model introduced by Goodfellow *et al*. Inspired by Szegedy *et al*.'s work on adversarial examples, Goodfellow *et al*. found that adversarial examples can be used to improve the representation of deep learning and conduct unsupervised learning [54]. [1] They generated samples using a generative network (generator) and used a discriminative network (discriminator) as an adversary to determine the generated samples are real or fake. This kind of network architectures is referred as Generative Adversarial Network (GAN).

The optimization function of GAN is described by:

$$\min_G \max_D V(D,G) = \mathop{\mathbb{E}}_{x\sim\mathcal{P}_r}[\log D(x)] + \mathop{\mathbb{E}}_{z\sim\mathcal{P}_z}[\log D(G(z))], \tag{4}$$

where D and G denote the discriminator and generator, $\mathcal{P}_r, \mathcal{P}_z$ are the distribution of input data and noise. In this competing fashion, GAN is capable of generating raw data samples that look close to the real data.

Wasserstein GAN (WGAN) stabilized the training of GAN by minimizing the objective function with Wasserstein-1 distance [56]:

$$W(\mathcal{P}_r, \mathcal{P}_z) = \max_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathcal{P}_r}[D(x)] - \mathbb{E}_{z\sim\mathcal{P}_z}[D(G(z))]. \tag{5}$$

The classifier of discriminator D satisfies $1 - Lipschitz$.

GAN generates data in an unsupervised way. GAN can be used to augment data samples, provide image super-resolution [57], and image-to-image translation [58], [59]. The discriminator networks can be used in the supervised learning as well.

AutoEncoder (AE) is another type of generative model composed of two parts: encoder $f_{enc}(x)$ and decoder $f_{dec}(z)$. Encoder maps input to a latent representation (low dimensional) z and decoder maps z back to a high-dimensional output $\hat{x}$. Both the encoder and the decoder are deep neural networks. Autoencoders are trained to minimize the difference between inputs and outputs. It can be used in model compression/decompression, unsupervised learning, etc.

2) General techniques used in deep learning: Besides CNNs and RNNs, many techniques have been discovered in deep learning. Most of recent deep learning models are built upon these techniques. Adversarial examples also leverage these techniques. We briefly discuss the following techniques.

Dropout is a simple regularization approach for deep neural networks by randomly removing hidden neurons in the training stage. Only part of the networks can learn from training dataset in each epoch. It can be seen as the ensemble of sub-networks. Dropout has been shown to avoid overfitting and make deep learning models more robust and generalized.

Activation functions are usually used to provide non-linearity in deep neural networks. Rectified linear unit (ReLU) function is a most commonly used activation function, which outputs zero when the unit is negative: $max(x, 0)$. Leaky rectified linear function (LeakyReLU) [60], parametric rectified linear unit (PReLU) [50] allow a modified output when the

[1]The timeline of these research projects according to www.quora.com/In-what-way-are-Adversarial-Networks-related-or-different-to-Adversarial\-Training: 1) Szegedy's [19]; 2) Goodfellow's GAN model [54]; 3) Goodfellow's [55].

unit is below zero. Scaled exponential linear unit (SELU) [61] introduce self-normalization in the activation function.

Batch Normalization is applied to normalize hidden units before applying activation function among batch samples during training. Similarly to normalization in the preprocessing, batch normalization rescales the hidden units and shifts them to their average value. An affine transformation is recommended at the end of batch normalization [62]. Batch normalization speeds up the training process in many applications, especially for neural networks with very deep layers.

Some techniques used in conventional machine learning algorithms are also effective in deep learning. Several regularization methods are applied during training to avoid overfitting: **Adding ℓ^1 or ℓ^2 regularization** term limits the weights of the neurons. **Early stopping**, a technique applied during the training stage before the loss increases, improves the performance outside the training dataset. To get the optimal parameters of neurons in each layer, many optimization approaches have been applied in deep learning, such as **Stochastic Gradient Descent** (SGD) with or without momentum, AdaGrad [63], RMSProp, ADAM [64].

3) Architectures of deep neural networks: Several deep learning architectures are widely used in computer vision tasks. *LeNet* [65], *VGG* [2], *AlexNet* [1], *GoogLeNet* [66]–[68] (Inception V1, Inception V2, Inception V3, and Inception V4), and *ResNet* [50] are widely used networks, from the simplest (oldest) network to the deepest and the most complex (newest) one. *AlexNet* first showed that deep learning models largely outperformed conventional machine learning algorithms based on the ImageNet challenge in 2012 and led the future study of deep learning. These architectures made tremendous breakthroughs in the ImageNet challenge and can be seen as the milestones in image classification problem. These popular architectures are referred as baselines and attackers usually generate adversarial examples against them.

4) General deep learning datasets: MNIST, *CIFAR10*, *ImageNet* are three widely used datasets in computer vision tasks. The *MNIST* dataset is for handwritten digits recognition (from 0 to 9) [69]. The *CIFAR10* dataset and the *ImageNet* dataset are for image recognition task [70]. The *CIFAR10* consists of 60,000 tiny color images (32x32) with 10 classes. The *ImageNet* dataset consists 14,197,122 images with 1,000 classes [71]. Because of the large number of images in the *ImageNet* dataset, most of the adversarial approaches are evaluated on only part of the *ImageNet* dataset. The *Street View House Numbers (SVHN)* dataset, similar to the *MNIST* dataset, consists of ten digits obtained from real-world house numbers in Google Street View images. *YoutubeDataset* is referred to the dataset gained from Youtube consisting about ten million images [65], which was used in [19].

B. Adversarial Examples and Countermeasures in Machine Learning

Adversarial examples against conventional machine learning models have been discussed since decades ago. Machine learning-based systems with handcrafted features are the main targets, such as spam filters, intrusion detection, biometric authentication, fraud detection, etc [72]. For example, spam emails are often modified to avoid detection [73]–[75].

Dalvi *et al*. first discussed adversarial examples and formulated this problem as a game between adversary and classifier (Naïve Bayes), both of which are sensitive to cost [73]. The attack and defense on adversarial examples became an iterative game. Biggio *et al*. first tried a gradient-based approach to generate adversarial examples against linear classifier, support vector machine (SVM), and neural network [76]. Compared with deep learning adversarial examples, their methods allow greater freedom to modify the data. The MNIST dataset was first evaluated under their attack, although a human could easily distinguish the adversarial digit images. Biggio *et al*. also reviewed several proactive defenses and discussed reactive approaches to improve the security of machine learning models [39].

Barreno *et al*. presented an initial investigation on the security problems of machine learning [72], [77]. They categorized attacking against machine learning system into three axes: 1) influence: whether attacks can poison the training data; 2) security violation: adversarial examples belong to false positive or false negative; 3) specificity: attack is targeted to a particular instance or a wide class. We discuss these axes for deep learning area in Section III. Barreno *et al*. compared attacks against SpamBayes spam filter as well defenses as a study case. However, they mainly focused on binary classification problem such as virus detection system, intrusion detection system (IDS), and intrusion prevention system (IPS).

Conventional machine learning (ML) adversarial examples require knowledge of feature selection, while deep learning (DL) usually needs only raw data input. In conventional ML, both attacking and defending methods paid great attention to features (even the previous step: data collection), giving less attention to the impact of humans. Then the target becomes a fully automatic machine learning system. Inspired by these papers on conventional ML, we review the recent security issues in the deep learning area in the paper.

[37] provided a comprehensive overview of security issues in machine learning and recent findings in deep learning. [37] established a unifying threat model. A "no free lunch" theorem was introduced: the tradeoff between accuracy and robustness. [37] covered a wide range of security problems. Compared to their work, our paper focuses on adversarial examples and have a detailed discussion on recent studies and findings.

III. Taxonomy of Adversarial Examples

We present a taxonomy to categorize the methods for generating adversarial examples along seven axes in this section.

- Adversarial Falsification
  - *False positive* attacks generate a negative sample which is misclassified as a positive one (Type I Error). In a malware detection task, a benign software being classified as malware is a false positive. In an image classification task, a false positive can be an adversarial image unrecognizable to human, but deep neural networks predict it to a class with high confidence. A false positive example of image classification is depicted in Figure 2.

  - *False negative* attacks generate a positive sample which is misclassified as a negative one (Type II Error). In a malware detection task, a false negative can be the condition that a malware (usually considered as positive) cannot be identified by the trained model. This is also called machine learning evasion. This error is shown in most adversarial images, where an image can be recognized by human, but the neural networks cannot identify it.
- Adversary's Knowledge
  - *White-box* attacks assume the adversary knows everything related to the trained neural network model: training data, network architectures, hyper-parameters, numbers of layers, functions of activations, network weights, etc. Many adversarial examples are generated by calculating network gradients. Since deep neural networks tend to require only raw input data without handcrafted features and to deploy end-to-end structure, feature selection is not necessary compared to adversarial examples in machine learning.
  - *Black-box* attacks assume the adversary has no access to the trained neural network model. The adversary, acting as a standard user, only knows the output of the model (label or confidence score). This assumption is common for attacking online Machine Learning services (*e.g.*, Machine Learning on AWS[2], Google Cloud AI[3], BigML[4], Microsoft Azure[6], IBM Bluemix[7], Face++[8]).
    Most adversarial example attacks are *white-box attacks*. However, they can be transferred to attack *black-box* services due to the transferability of adversarial examples proposed by Papernot *et al.* [44]. We will elaborate on it in Section VII-A. [78] and [79] are *black-box attacks* without transferability.
- Adversarial Specificity
  - *Targeted* attacks misguide the deep neural networks to a specific class. Targeted attacks usually occur in the multi-class classification problem. For example, an adversary fools an image classifier to predict all the adversarial examples as one class. In a face recognition/biometric system, an adversary tries to disguise a face as an authorized user (Impersonation) [80]. Targeted attacks usually maximize the probability of the targeted adversarial class.
  - *Non-targeted* attacks do not assign a specific class to the neural network output. The adversarial class of output can be arbitrary except the original one. Non-targeted attacks are easier to implement compared to *targeted* attacks since it has more options and space to redirect the output. For example, an adversary makes his/her face misidentified as an arbitrary face in face recognition system to evade detection (Dodging) [80]. Non-targeted adversarial examples are usually generated in two ways: 1) running several targeted attacks and taking the one with the smallest perturbation from the results; 2) minimizing the probability of the correct class.
    Some generating approaches (*e.g.*, extended BIM, ZOO) can be applied to both targeted and non-targeted attacks. For binary classification, *targeted* attacks are equivalent to *non-targeted* attacks.
- Perturbation Scope
  - *Individual* attacks generate different perturbations for each clean input.
  - *Universal* attacks only create a universal perturbation for the whole dataset. This perturbation can be applied to all clean input data.
    Most of the current attacks generate adversarial examples individually. However, *universal* perturbations make it easier to deploy adversary examples in the real world. Adversaries do not require to change the perturbation when the input sample changes.
- Perturbation Limitation
  - *Optimized Perturbation* sets perturbation as the objective of the optimization problem. These methods aim to minimize the perturbation, so that humans cannot recognize the perturbation.
  - *Constraint Perturbation* sets perturbation as the constraint of the optimization problem. These methods only require the perturbation small enough not to be recognized by a human.
- Attack Frequency
  - *One-time Attacks* take only one time to generate adversarial examples.
  - *Iterative Attacks* take multiple times to update the adversarial examples.
    Compared with *one-time attacks*, *iterative attacks* usually perform better adversarial examples, but require more interactions with victim classifier (posting more queries) and cost more computational time to generate them. For some computational-intensive tasks (*e.g.*, Reinforcement Learning), *one-time attacking* may be the only feasible choice.
- Perturbation Measurement
  - ℓ_p measures the magnitude of perturbation by p-norm distance:

$$\|x\|_p = \left(\sum_{i=1}^{n} \|x_i\|^p\right)^{\frac{1}{p}}. \tag{6}$$

    $\ell_0, \ell_2, \ell_\infty$ are three commonly used ℓ_p metrics. ℓ_0 counts the number of pixels changed in the adversarial examples; ℓ_2 measures the Euclidean distance between the adversarial example and the original sample; ℓ_∞ denotes the maximum changes for all pixels in adversarial examples.
  - *Psychometric perceptual adversarial similarity score (PASS)* is a new metric introduced in [81], consistent with human perception.

In the following sections, we will investigate recent studies on adversarial examples according to this taxonomy.

[2] https://aws.amazon.com/machine-learning
[3] https://cloud.google.com/products/machine-learning
[4] https://bigml.com, Clarifai[5]
[6] https://azure.microsoft.com/en-us/services/machine-learning
[7] https://console.bluemix.net/catalog/services/machine-learning
[8] https://www.faceplusplus.com

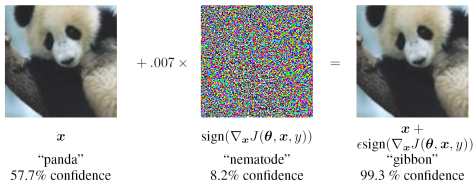

Figure 1: An adversarial image generated by *Fast Gradient Sign Method* [55]: left: a clean image of a panda; middle: perturbation; right: an adversarial image, classified as a gibbon.

IV. Methods for generating Adversarial Examples

In this section, we illustrate several representative approaches for generating adversarial examples. Although many of these approaches are defeated by a countermeasure in later studies, we present these methods to show how the adversarial examples generating approaches improved and to what extent current state-of-the-art adversarial examples attacks can achieve. The existence of these methods also requires investigation, which may improve the robustness of deep neural networks. Table II presents these methods along seven axes discussed in taxonomy.

A. L-BFGS Attack

Szegedy *et al*. first introduced adversarial examples for deep neural networks in 2014 [19]. They generated adversarial examples using a L-BFGS method to solve the general targeted problem:

$$\begin{aligned} \min_{x'} \quad & c\|\eta\| + J_\theta(x', l') \\ s.t. \quad & x' \in [0, 1]. \end{aligned} \tag{7}$$

To find a suitable constant c, *L-BFGS Attack* calculated approximate values of *Adversarial Examples* by line-searching $c > 0$. The author showed that the generated adversarial examples could also be generalized to different models and different training datasets. They suggested that adversarial examples are due to never/rarely seen examples in the test datasets.

L-BFGS Attack was also used in [91], which implemented a binary search to find the optimal value for c.

B. Fast Gradient Sign Method (FGSM)

Because *L-BFGS Attack* used an expensive linear search method to find the optimal value, which was time-consuming and impractical. Goodfellow *et al*. proposed a fast method for generating adversarial examples called *Fast Gradient Sign Method* [55]. They only performed one step gradient update along the direction of the sign of gradient at each pixel. Their perturbation can be expressed as:

$$\eta = \epsilon sign(\nabla_x J_\theta(x, l)), \tag{8}$$

where ϵ is the magnitude of the perturbation. Thus The generated adversarial example x' is calculated as: $x' = x + \eta$. This perturbation can be computed simply using backpropagation. Figure 1 shows an adversarial example on ImageNet.

They claimed that the linear part of the high dimensional deep neural network could not resist adversarial examples, although linear behavior speeded up training. Regularization approaches used in deep neural networks such as dropout, pretraining could not improve the robustness of networks.

[81] proposed a new method, called *Fast Gradient Value* method, in which they replaced the sign of the gradient with the raw gradient: $\eta = \nabla_x J(\theta, x, l)$. *Fast Gradient Value* method has no constraints in each pixel and can generate images with a larger local difference.

According to [92], one-step attack is easy to transfer but also easy to defend (see Section VII-A). [93] applied momentum to *FGSM* to generate adversarial examples in a more iterative way. The gradient and adversarial examples were calculated by:

$$\begin{aligned} \mathbf{g}_{t+1} &= \mu \mathbf{g}_t + \frac{\nabla_x J_\theta(x'_t, l)}{\|\nabla_x J_\theta(x'_t, l)\|}, \\ x'_{t+1} &= x'_t + \epsilon sign \mathbf{g}_{t+1} \end{aligned} \tag{9}$$

The authors increased the effectiveness of attack by introducing momentum and improved the transferability by applying the one-step attack and the ensembling method. This method won NIPS 2017 targeted and non-targeted adversarial attacks competition.

[92] extended *FGSM* to a targeted attack by maximizing the probability of the target class:

$$x' = x - \epsilon sign(\nabla_x J(\theta, x, l')). \tag{10}$$

The authors refer this attack to *One-step Target Class Method (OTCM)*.

[94] found that *FGSM* with adversarial training is more robust to white-box attacks than to black-box attacks due to *gradient masking*. They proposed a new attack, *RAND-FGSM*, which added random when updating the adversarial examples to defeat adversarial training:

$$\begin{aligned} x_{tmp} &= x + \alpha \cdot sign(\mathcal{N}(\mathbf{0}^d, \mathbf{I}^d)), \\ x' &= x_{tmp} + (\epsilon - \alpha) \cdot sign(\nabla_{x_{tmp}} J(x_{tmp}, l)), \end{aligned} \tag{11}$$

where α, ϵ are the parameters, $\alpha < \epsilon$.

C. Basic Iterative Method (BIM) and Iterative Least-Likely Class Method (ILLC)

Previous methods assume adversarial data can be directly fed into deep neural networks. However, in many applications, people can only pass data through devices (*e.g*., cameras, sensors). Kurakin *et al*. applied adversarial examples to the physical world [20]. They extended *Fast Gradient Sign* method by running a finer optimization (smaller change) for multiple iterations. In each iteration, they clipped pixel values to avoid large change on each pixel:

$$Clip_{x,\xi}\{x'\} = \min\{255, x + \xi, \max\{0, x - \epsilon, x'\}\}, \tag{12}$$

where $Clip_{x,\xi}\{x'\}$ is the clipping value in each iteration limited by ξ. The adversarial examples were generated in multiple iterations:

$$\begin{aligned} x_0 &= x, \\ x_{n+1} &= Clip_{x,\xi}\{x_n + \epsilon sign(\nabla_x J(x_n, y))\}. \end{aligned} \tag{13}$$

Authors referred to this method as *Basic Iterative* method.

To further attack a specific class, they chose the least-likely class of the prediction and try to maximize the cross-entropy

Table II: Taxonomy of Adversarial Examples

Attacks Methods	Adversarial Falsification	Adversary's Knowledge	Adversarial Specificity	Perturbation Scope	Perturbation Limitation	Attack Frequency	Perturbation Measurement	Datasets	Architectures
L-BFGS Attack [19]	False Negative	White-Box	Targeted	Individual	Optimized	Iterative	ℓ_2	MNIST, ImageNet, YoutubeDataset	AlexNet, QuocNet
Fast Gradient Sign Method (FGSM) [55]	False Negative	White-Box	Non-Targeted	Individual	N/A	One-time	element-wise	MNIST, ImageNet	GoogLeNet
Basic Iterative Method (BIM) and Iterative Least-Likely Class (ILLC) [20]	False Negative	White-Box	Non-Targeted	Individual	N/A	Iterative	element-wise	ImageNet	GoogLeNet
Jacobian-based Saliency Map Attack (JSMA) [82]	False Negative	White-Box	Targeted	Individual	Optimized	Iterative	ℓ_2	MNIST	LeNet
DeepFool [83]	False Negative	White-Box	Non-Targeted	Individual	Optimized	Iterative	$\ell_p(p \in 1, \infty)$	MNIST, CIFAR10, ImageNet	LeNet, CaffeNet, GoogLeNet
CPPN EA Fool [84]	False Positive	White-Box	Non-Targeted	Individual	N/A	Iterative	N/A	MNIST, ImageNet	LeNet, AlexNet
C&W's Attack [85]	False Negative	White-Box	Targeted	Individual	Optimized	Iterative	$\ell_1, \ell_2, \ell_\infty$	MNIST, CIFAR10, ImageNet	GoogLeNet
Zeroth Order Optimization [78]	False Negative	Black-Box	Targeted & Non-Targeted	Individual	Optimized	Iterative	ℓ_2	CIFAR10, ImageNet	GoogLeNet
Universal Perturbation [86]	False Negative	White-Box	Non-Targeted	Universal	Optimized	Iterative	$\ell_p(p \in 1, \infty)$	ImageNet	CaffeNet, VGG, GoogLeNet, ResNet
One Pixel Attack [87]	False Negative	Black-Box	Targeted & Non-Targeted	Individual	Constraint	Iterative	ℓ_0	CIFAR10	VGG, AllConv, NiN
Feature Adversary [88]	False Negative	White-Box	Targeted	Individual	Constraint	Iterative	ℓ_2	ImageNet	CaffeNet, VGG, AlexNet, GoogLeNet
Hot/Cold [81]	False Negative	White-Box	Targeted	Individual	Optimized & Constraint	One-time	PASS	MNIST, ImageNet	LeNet, GoogLeNet, ResNet
Natural GAN [79]	False Negative	Black-Box	Non-targeted	Individual	Optimized	Iterative	ℓ_2	MNIST, LSUN, SNLI	LeNet, LSTM, TreeLSTM
Model-based Ensembling Attack [89]	False Negative	White-Box	Targeted & Non-Targeted	Individual	Constraint	Iterative	ℓ_2	ImageNet	VGG, GoogLeNet, ResNet
Ground-Truth Attack [90]	False Negative	White-Box	Targeted	Individual	Optimized	Iterative	ℓ_1, ℓ_∞	MNIST	3-layer FC

loss. This method is referred to *Iterative Least-Likely Class* method:

$$\begin{aligned} x_0 &= x, \\ y_{LL} &= arg\min_y\{p(y|x)\}, \\ x_{n+1} &= Clip_{x,\epsilon}\{x_n - \epsilon sign(\nabla_x J(x_n, y_{LL}))\}. \end{aligned} \tag{14}$$

They successfully fooled the neural network with a crafted image taken from a cellphone camera. They also found that *Fast Gradient Sign* method is robust to phototransformation, and iterative methods cannot resist phototransformation.

D. Jacobian-based Saliency Map Attack (JSMA)

Papernot *et al*. designed an efficient saliency adversarial map, called *Jacobian-based Saliency Map Attack* [82]. They first computed Jacobian matrix of given sample x, which is given by [9]:

$$J_f(x) = \frac{\partial f(x)}{\partial x} = \left[\frac{\partial f_j(x)}{\partial x_i}\right]_{i\times j}. \tag{15}$$

In this way, they found the input features of x that made most significant changes to the output. A small perturbation was designed to successfully induce large output variations so that change in a small portion of features could fool the neural network.

Then authors defined two adversarial saliency maps to select the feature/pixel to be crafted in each iteration. They achieved 97% adversarial success rate by modifying only 4.02% input features per sample. However, this method runs very slow due to its significant computational cost.

E. DeepFool

Moosavi-Dezfooli *et al*. proposed *DeepFool* to find the closest distance from original input to the decision boundary of adversarial examples [83]. To overcome the non-linearity in high dimension, they performed an iterative attack by linear approximation. Starting from an affine classifier, they found that the minimal perturbation of an affine classifier is the distance to the separating affine hyperplane $\mathcal{F} = \{x : w^T x + b = 0\}$. The perturbation of an affine classifier f can be $\eta^*(x) = -\frac{f(x)}{\|w\|^2}w$.

If f is a binary differentiable classifier, they used an iterative method to approximate the perturbation by considering f is linearized around x_i at each iteration. The minimal perturbation is computed as:

$$\begin{aligned} \arg\min_{\eta_i} \quad & \|\eta_i\|_2 \\ s.t. \quad & f(x_i) + \nabla f(x_i)^T \eta_i = 0. \end{aligned} \tag{16}$$

This result can also be extended to the multi-class classifier by finding the closest hyperplane. It can also be extended to more general ℓ_p norm, $p \in [0, \infty)$. *DeepFool* provides less perturbation compared to *FGSM* and *JSMA*. Compared to *JSMA*, *DeepFool* also reduced the intensity of perturbation instead of the number of selected features.

[9] As mentioned in [85], Papernot *et al*. do not use last softmax layer when calculating Jacobian matrix. Carlini and Wagner modify this approach by adding the last layer in [85].

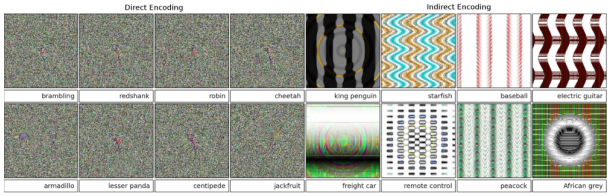

Figure 2: Unrecognizable examples to humans, but deep neural networks classify them to a class with high certainty ($\geq 99.6\%$) [84]

F. CPPN EA Fool

Nguyen *et al*. discovered a new type of attack, compositional pattern-producing network-encoded EA (CPPN EA), where adversarial examples are classified by deep neural networks with high confidence (99%), however, unrecognizable to human [84]. We categorize this kind of attack as a *False positive* attack. Figure 2 illustrates such kind of adversarial examples.

They used evolutionary algorithms (EAs) algorithm to produce the adversarial examples. To solve multi-class classification problem using EA algorithms, they applied multi-dimensional archive of phenotypic elites MAP-Elites [95]. Authors first encoded the images with two different methods: direct encoding (grayscale or HSV value) and indirect encoding (compositional pattern-producing network). Then in each iteration, MAP-Elites, like general EA algorithm, chose a random organism, mutated it randomly, and replaced with the current one if the new one has higher *fitness* (high certainty for a class of a neural network). In this way, MAP-Elites can find the best individual for each class. As they claimed, for many adversarial images, CPPN could easily locate the important features to change the output of deep neural networks just like JSMA did. Many images from same evolutionary are found similar for closely related categories. More interestingly, CPPN EA fooling images are accepted by an art contest with 35.5% acceptance rate.

G. C&W's Attack

Carlini and Wagner launched a targeted attack to defeat *Defensive distillation* (Section VI-A) [85]. According to their further study [96], [97], *C&W's Attack* is effective for most of existing adversarial detecting defenses. The authors made several modifications based on the basic problem (Equation 1).

They first defined a new objective function g, so that:

$$\begin{aligned} \min_{\eta} \quad & \|\eta\|_p + c \cdot g(x+\eta) \\ s.t. \quad & x + \eta \in [0,1]^n, \end{aligned} \tag{17}$$

where $g(x') \geq 0$ if and only if $f(x') = l'$. In this way, the distance and penalty term can be better optimized. The authors listed seven example objective functions g. An effective function evaluated by their experiments can be:

$$g(x') = \max(\max_{i \neq l'}(Z(x')_i) - Z(x')_t, -\kappa), \tag{18}$$

where Z presents the Softmax function, κ is a constant to control the confidence (κ is set as 0 in [85]).

Second, instead of using box-constrained L-BFGS to find minimal perturbation in *L-BFGS Attack* method, the authors

introduced a new variant w to avoid the box constraint, where w satisfies $\eta = \frac{1}{2}(\tanh(w)+1)-x$. General optimizers in deep learning like Adam and SGD were used to generate adversarial examples and performed 20 iterations of such generation to find an optimal c by binary searching. However, they found that if the gradients of $\|\eta\|_p$ and $g(x+\eta)$ are not in the same scale, it is hard to find a suitable constant c in all of the iterations of the gradient search and then get the optimal result. Due to this reason, two of their proposed functions did not find optimal solutions for adversarial examples.

Third, three distance measurements of perturbation were discussed in the paper: ℓ_0 attack, ℓ_2, and ℓ_∞. The authors provided three kinds of attacks based on the distance metrics: ℓ_0 attack, ℓ_2 attack, and ℓ_∞ attack.

ℓ_2 attack can be described by:

$$\min_w \|\frac{1}{2}(\tanh(w)+1)\|_2 + c \cdot g(\frac{1}{2}\tanh(w)+1). \quad (19)$$

The author showed that distillation network could not help defend ℓ_2 attack.

ℓ_0 attack was conducted iteratively because ℓ_0 is not differentiable. In each iteration, a few pixels are considered trivial for generating adversarial examples and removed. The importance of pixels is determined by the gradient of ℓ_2 distance. The iteration stops if the remaining pixels can not generate an adversarial example.

ℓ_∞ attack was also an iterative attack, which replaced the ℓ_2 term with a new penalty in each iteration:

$$\min \quad c \cdot g(x+\eta) + \sum_i [(\eta_i - \tau)^+]. \quad (20)$$

For each iteration, they reduced τ by a factor of 0.9, if all $\eta_i < \tau$. ℓ_∞ attack considered τ as an approximate measurement of ℓ_∞.

H. Zeroth Order Optimization (ZOO)

Different from gradient-based adversarial generating approaches, Chen *et al*. proposed a *Zeroth Order Optimization (ZOO)* based attack [78]. Since this attack does not require gradients, it can be directly deployed in a black-box attack without transferability. Inspired by [85], the authors modified $g(\cdot)$ in [85] as a new hinge-like loss function:

$$g(x') = \max(\max_{i \neq l'}(\log[f(x)]_i) - \log[f(x)]_{l'}, -\kappa), \quad (21)$$

and used symmetric difference quotient to estimate the gradient and Hessian:

$$\begin{aligned} \frac{\partial f(x)}{\partial x_i} &\approx \frac{f(x+he_i) - f(x-he_i)}{2h}, \\ \frac{\partial^2 f(x)}{\partial x_i^2} &\approx \frac{f(x+he_i) - 2f(x) + f(x-he_i)}{h^2}, \end{aligned} \quad (22)$$

where e_i denotes the standard basis vector with the ith component as 1, h is a small constant.

Through employing the gradient estimation of gradient and Hessian, *ZOO* does not need the access to the victim deep learning models. However, it requires expensive computation to query and estimate the gradients. Based on stochastic coordinate descent (SCD) algorithms, the authors proposed ADAM like algorithms, *ZOO-ADAM*, to randomly select a variable and update the adversarial examples. The experiments showed that *ZOO* achieved the comparable performance as *C&W's Attack*.

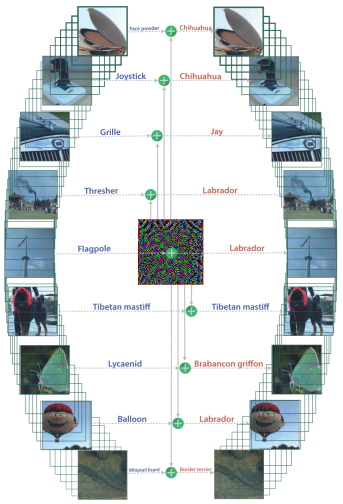

Figure 3: A universal adversarial example fools the neural network on images. Left images: original labeled natural images; center image: universal perturbation; right images: perturbed images with wrong labels. [86]

I. Universal Perturbation

Leveraging their previous method on *DeepFool*, Moosavi-Dezfooli *et al*. developed a universal adversarial attack [86]. The problem they formulated is to find a universal perturbation vector satisfying

$$\begin{aligned} \|\eta\|_p &\leq \epsilon \\ \mathcal{P}\big((x') \neq f(x)\big) &\geq 1-\delta. \end{aligned} \quad (23)$$

ϵ limits the size of universal perturbation, and δ controls the failure rate of all the adversarial samples.

For each iteration, they use *DeepFool* method to get a minimal sample perturbation against each input data and update the perturbation to the total perturbation η. This loop will not stop until most data samples are fooled ($\mathcal{P} < 1-\delta$). From the experiments, the universal perturbation can be generated by using a small part of data samples instead of the entire dataset. Figure 3 illustrates a universal adversarial example can fool a group of images. The universal perturbations were shown to be generalized well across popular deep learning architectures (*e.g*., VGG, CaffeNet, GoogLeNet, ResNet).

J. One Pixel Attack

To avoid the problem of measurement of perceptiveness, Su *et al*. generated adversarial examples by only modifying one pixel [87]. The optimization problem becomes:

$$\begin{aligned} \min_{x'} \quad & J(f(x'), l') \\ s.t. \quad & \|\eta\|_0 \leq \epsilon_0, \end{aligned} \quad (24)$$

where $\epsilon_0 = 1$ for modifying only one pixel. The new constraint made it hard to optimize the problem.

Su *et al*. applied differential evolution (DE), one of the evolutionary algorithms, to find the optimal solution. DE does

not require the gradients of the neural networks and can be used in non-differential objective functions. They evaluated the proposed method on the *CIFAR10* dataset using three neural networks: All convolution network (AllConv) [98], Network in Network (NiN) [99], and VGG16. Their results showed that 70.97% of images successfully fooled deep neural networks with at least one target class with confidence 97.47% on average.

K. Feature Adversary

Sabour *et al*. performed a targeted attack by minimizing the distance of the representation of internal neural network layers instead of the output layer [88]. We refer to this attack as *Feature Adversary*. The problem can be described by:

$$\begin{aligned} \min_{x'} \quad & \|\phi_k(x) - \phi_k(x')\| \\ s.t. \quad & \|x - x'\|_\infty < \delta, \end{aligned} \tag{25}$$

where ϕ_k denotes a mapping from image input to the output of the kth layer. Instead of finding a minimal perturbation, δ is used as a constraint of perturbation. They claimed that a small fixed value δ is good enough for human perception. Similar to [19], they used L-BFGS-B to solve the optimization problem. The adversarial images are more natural and closer to the targeted images in the internal layers.

L. Hot/Cold

Rozsa *et al*. proposed a *Hot/Cold* method to find multiple adversarial examples for every single image input [81]. They thought small translations and rotations should be allowed as long as they were *imperceptible*.

They defined a new metric, Psychometric Perceptual Adversarial Similarity Score (PASS), to measure the noticeable similarity to humans. *Hot/Cold* ignored the unnoticeable difference based on pixels and replaced widely used ℓ_p distance with PASS. PASS includes two stages: 1) aligning the modified image with the original image; 2) measuring the similarity between the aligned image and the original one.

Let $\phi(x', x)$ be a homography transform from the adversarial example x' to the original example x. $\mathcal{H}$ is the homography matrix, with size 3×3. $\mathcal{H}$ is solved by maximizing the *enhanced correlation coefficient (ECC)* [100] between x' and x. The optimization function is:

$$\arg\min_{\mathcal{H}} \left\| \frac{\overline{x}}{\|\overline{x}\|} - \frac{\overline{\phi(x', x)}}{\|\overline{\phi(x', x)}\|} \right\|, \tag{26}$$

where $\bar{\cdot}$ denotes the normalization of an image.

Structural SIMilarity (SSIM) index [101] was adopted to measure the just noticeable difference of images. [81] leveraged SSIM and defined a new measurement, regional SSIM index (RSSIM) as:

$$RSSIM(x_{i,j}, x'_{i,j}) = L(x_{i,j}, x'_{i,j})^\alpha C(x_{i,j}, x'_{i,j})^\beta S(x_{i,j}, x'_{i,j})^\gamma,$$

where α, β, γ are weights of importance for luminance ($L(\cdot,\cdot)$), contrast ($C(\cdot,\cdot)$), and structure ($S(\cdot,\cdot)$). The SSIM can be calculated by averaging RSSIM:

$$SSIM(x_{i,j}, x'_{i,j}) = \frac{1}{n \times m} \sum_{i,j} RSSIM(x_{i,j}, x'_{i,j}).$$

PASS is defined by combination of the alignment and the similarity measurement:

$$PASS(x', x) = SSIM(\phi^*(x', x), x). \tag{27}$$

The adversarial problem with the new distance is described as:

$$\begin{aligned} \min \quad & D(x, x') \\ s.t. \quad & f(x') = y', \\ & PASS(x, x') \geq \gamma. \end{aligned} \tag{28}$$

where $D(x, x')$ is a measure of distance, which can be $1 - PASS(x, x')$ or $\|x - x'\|_p$.

To generate a diverse set of adversarial examples, authors defined targeted label l' as *hot* class, and original label l as *cold* class. In each iteration, they moved toward a target (hot) class while moving away from the original (cold) class. Their results showed that their generated adversarial examples are comparable to *FGSM*, and with more diversity.

M. Natural GAN

Zhao *et al*. utilized Generative Adversarial Networks (GANs) as part of their approach to generate adversarial examples of images and texts [79], which made adversarial examples more natural to human. We name this approach *Natural GAN*. The authors first trained a WGAN model on the dataset, where the generator $\mathcal{G}$ maps random noise to the input domain. They also trained an "inverter" $\mathcal{I}$ to map input data to dense inner representations. Hence, the adversarial noise was generated by minimizing the distance of the inner representations like "Feature Adversary." The adversarial examples were generated using the generator: $x' = \mathcal{G}(z')$:

$$\begin{aligned} \min_{z} \quad & \|z - \mathcal{I}(x)\| \\ s.t. \quad & f(\mathcal{G}(z)) \neq f(x). \end{aligned} \tag{29}$$

Both the generator $\mathcal{G}$ and the inverter $\mathcal{I}$ were built to make adversarial examples natural. *Natural GAN* was a general framework for many deep learning fields. [79] applied *Natural GAN* to image classification, textual entailment, and machine translation. Because *Natural GAN* does not require gradients of original neural networks, it can also be applied to *Black-box Attack*.

N. Model-based Ensembling Attack

Liu *et al*. conducted a study of transferability (Section VII-A) over deep neural networks on ImageNet and proposed a *Model-based Ensembling Attack* for targeted adversarial examples [89]. Authors argued that compared to non-targeted adversarial examples, targeted adversarial examples are much harder to transfer over deep models. Using *Model-based Ensembling Attack*, they can generate transferable adversarial examples to attack a black-box model.

The authors generated adversarial examples on multiple deep neural networks with full knowledge and tested them on a black-box model. *Model-based Ensembling Attack* was described as the following optimization problem:

$$\arg\min_{x'} \quad -\log\left((\sum_{i=1}^{k} \alpha_i J_i(x', l')) \right) + \lambda \|x' - x\|, \tag{30}$$

where k is the number of deep neural networks in the generation, f_i is the function of each network, and α_i is the ensemble weight, $\sum_i^k \alpha_i = 1$. The results showed that *Model-based Ensembling Attack* could generate transferable targeted adversarial images, which enhanced the power of adversarial examples for black-box attacks. The results also proved that this method performs better in generating non-targeted adversarial examples than previous methods. Authors successfully conducted a black-box attack against Clarifai.com using *Model-based Ensembling Attack*.

O. Ground-Truth Attack

Formal verification techniques aim to evaluate the robustness of a neural network even against zero-day attacks (Section VI-F). Carlini *et al*. constructed a ground-truth attack, which provided adversarial examples with minimal perturbation to the ground truth [90]. *Network Verification* always checks whether an adversarial example violates a property of a deep neural network. *Ground-Truth Attack* conducted a binary search and found adversarial examples within small perturbation that did not hold the property of the network.

V. Applications for Adversarial Examples

We have investigated adversarial examples for image classification task. In this section, we review adversarial examples against the other tasks. We mainly focus on three questions: What scenarios are adversarial examples applied in new tasks? How to generate adversarial examples for new tasks? Whether to propose a new method or to translate the problem into image classification task and solve it by the aforementioned methods? Table III summarizes the applications for adversarial examples in this section.

A. Reinforcement Learning

Deep neural networks have been used in reinforcement learning by training policies on raw input (*e.g*., images). [102], [103] generated adversarial examples on deep reinforcement learning policies. Since the inherent intensive computation of reinforcement learning, both of them performed fast *One-time attack*.

Huang *et al*. applied *FGSM* to attacks on three deep reinforcement learning models [102]: deep Q network (DQN), trust region policy optimization(TRPO), and asynchronous advantage actor-critic (A3C) [115]. Similarly to [55], they added small perturbations on the input of policy by calculating the gradient of the cross-entropy loss function: $\nabla_x J(\theta, x, \ell))$. Because DQN does not have stochastic policy input, softmax of Q-values is considered to calculate the loss function. They evaluated adversarial examples on four Atari 2600 games with three norm constraints $\ell_1, \ell_2, \ell_\infty$. They found *Huang's Attack* with ℓ_1 norm conducted a successful attack for not only white-box attack but also *Black-box attack* (no access to the training algorithms, parameters, and hyper-parameters).

[103] used *FGSM* to attack A3C algorithm and Atari Pong task. [103] found that injecting perturbations in a fraction of frames is sufficient.

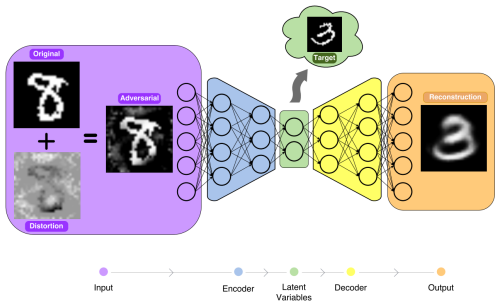

Figure 4: Adversarial attacks for autoencoders [105]. Perturbations are added to the input the encoder. After encoding and decoding, decoder will output an adversarial image presenting an incorrect class

B. Generative Modeling

Kos *et al*. [104] and Tabacof *et al*. [105] proposed adversarial examples for generative models. An adversary for autoencoder can inject perturbations into the input of encoder and generate a targeted class after decoding. Figure 4 depicts a targeted adversarial examples for an autoencoder. Adding perturbations on the input image of the encoder can misguide the autoencoder by making decoder to generating a targeted adversarial output image.

Kos *et al*. described a scenario to apply adversarial examples against autoencoder. Autoencoders can be used to compress data by an encoder and decompress by a decoder. For example, Toderici *et al*. use RNN-based AutoEncoder to compress image [116]. Ledig *et al*. used GAN to super-resolve images [57]. Adversaries can leverage autoencoder to reconstruct an adversarial image by adding perturbation to the input of the encoder.

Tabacof *et al*. used *Feature Adversary* Attack against AE and VAE. The adversarial examples were formulated as follows [105]:

$$\begin{aligned} \min_{\eta} \quad & D(z_{x'}, z_x) + c\|\eta\| \\ s.t. \quad & x' \in [L, U] \\ & z_{x'} = Encoder(x') \\ & z_x = Encoder(x), \end{aligned} \tag{31}$$

where $D(\cdot)$ is the distance between latent encoding representation z_x and $z_{x'}$. Tabacof *et al*. chose KL-divergence to measure $D(\cdot)$ in [105]. They tested their attacks on the MNIST and SVHN dataset and found that generating adversarial examples for autoencoder is much harder than for classifiers. VAE is even slightly more robust than deterministic autoencoder.

Kos *et al*. extended Tabacof *et al*.'s work by designing another two kinds of distances. Hence, the adversarial examples can be generated by optimizing:

$$\min_{\eta} \quad c\|\eta\| + J(x', l'). \tag{32}$$

The loss function J can be cross-entropy (refer to "Classifier Attack" in [104]), VAE loss function ("L_{VAE} Attck"), and distance between the original latent vector z and modified encoded vector x' ("Latent Attack", similar to Tabacof *et al*.'s work [105]). They tested VAE and VAE-GAN [117] on the MNIST, SVHN, and CelebA datasets. In their experimental results, "Latent Attack" achieved the best result.

Table III: Summary of Applications for Adversarial Examples

Applications	Representative Study	Method	Adversarial Falsification	Adversary's Knowledge	Adversarial Specificity	Perturbation Scope	Perturbation Limitation	Attack Frequency	Perturbation Measurement	Dataset	Architecture
Reinforcement Learning	[102]	FGSM	N/A	White-box & Black-box	Non-Targeted	Individual	N/A	One-time	$\ell_1, \ell_2, \ell_\infty$	Atari	DQN, TRPO, A3C
	[103]	FGSM	N/A	White-box	Non-Targeted	Individual	N/A	One-time	N/A	Atari Pong	A3C
Generative Modeling	[104]	Feature Adversary, C&W	N/A	White-box	Targeted	Individual	Optimized	Iterative	ℓ_2	MNIST, SVHN, CelebA	VAE, VAE-GAN
	[105]	Feature Adversary	N/A	White-box	Targeted	Individual	Optimized	Iterative	ℓ_2	MNIST, SVHN	VAE, AE
Face Recognition	[80]	Impersonation & Dodging Attack	False negative	white-box & black-box	Targeted & Non-Targeted	Universal	Optimized	Iterative	Total Variation	LFW,	VGGFace
Object Detection	[22]	DAG	False negative & False positive	White-box & Black-box	Non-Targeted	Individual	N/A	Iterative	N/A	VOC2007, VOC2012	Faster-RCNN
Semantic Segmentation	[22]	DAG	False negative & False positive	White-box & Black-box	Non-Targeted	Individual	N/A	Iterative	N/A	DeepLab	FCN
	[106]	ILLC	False negative	White-box	Targeted	Individual	N/A	Iterative	ℓ_∞	Cityscapes	FCN
	[107]	ILLC	False negative	White-box	Targeted	Universal	N/A	Iterative	N/A	Cityscapes	FCN
Reading Comprehension	[108]	AddSent, AddAny	N/A	Black-box	Non-Targeted	Individual	N/A	One-time & Iterative	N/A	SQuAD	BiDAF, Match-LSTM, and twelve other published models
	[109]	Reinforcement Learning	False negative	White-box	Non-Targeted	Individual	Optimized	Iterative	ℓ_0	TripAdvisor Dataset	Bi-LSTM, memory network
Malware Detection	[110]	JSMA	False negative	White-box	Targeted	Individual	Optimized	Iterative	ℓ_2	DREBIN	2-layer FC
	[111]	Reinforcement Learning	False negative	Black-box	Targeted	Individual	N/A	Iterative	N/A	N/A	Gradient Boosted Decision Tree
	[112]	GAN	False negative	Black-box	Targeted	Individual	N/A	Iterative	N/A	malwr	Multi-layer Perceptron
	[113]	GAN	False negative	Black-box	Targeted	Individual	N/A	Iterative	N/A	Alexa Top 1M	Random Forest
	[114]	Generic Programming	False negative	Black-box	Targeted	Individual	N/A	Iterative	N/A	Contagio	Random Forest, SVM

Figure 5: An example of adversarial eyeglass frame against Face Recognition System [80]

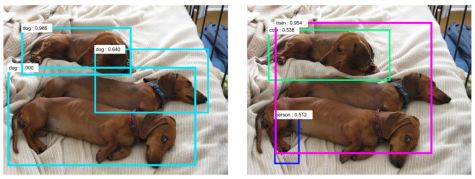

Figure 6: An adversarial example for object detection task [22]. Left: object detection on a clean image. Right: object detection on an adversarial image.

C. Face Recognition

Deep neural network based Face Recognition System (FRS) and Face Detection System have been widely used due to its high performance. [80] first provided a design of eyeglass frames to attack a deep neural network based FRS [118], which composes 11 blocks with 38 layers and *triplet loss* function of embedding features. Based on the *triplet loss* function, [80] designed a *softmaxloss* function:

$$J(x) = -\log\left(\frac{e^{<h_{c_x}, f(x)>}}{\sum_{c=1}^{m} e^{<h_c, f(x)>}}\right), \tag{33}$$

where h_c is a one-hot vector of class c, $< \cdot, \cdot >$ denotes inner product. Then they used *L-BFGS Attack* to generate adversarial examples.

In a further step, [80] implemented adversarial eyeglass frames to achieve attack in the physical world: the perturbations can only be injected into the area of eyeglass frames. They also enhanced the printability of adversarial images on the frame by adding a penalty of non-printability score (NPS) to the optimized objective. Similarly to *Universal Perturbation*, they optimize the perturbation to be applied to a set of face images. They successfully dodged (non-targeted attack) against FRS (over 80 % time) and also misguided FRS as a specific face (targeted attack) with a high success rate (depending on the target). Figure 5 illustrates an example of adversarial eyeglass frames.

Leveraging the approach of printability, [21] proposed an attack algorithm, Robust Physical Perturbations (RP_2), to modify the road sign (stop sign to speed limit sign) [10]. They changed the physical road signs by two kinds of attacks: 1) overlaying an adversarial road sign over a physical sign; 2) sticking perturbations on an existing sign. [21] included the non-printability score in the optimization objective to improve the printability.

D. Object Detection

The object Detection task is to find the proposal of an object (bounding box), which can be viewed as an image classification task for every possible proposal. [22] proposed a universal algorithm called *Dense Adversary Generation (DAG)* to generate adversarial examples for both object detection and semantic segmentation. The authors aimed at making the prediction (detection/segmentation) incorrect (non-targeted). Figure 6 illustrates an adversarial example for the object detection task.

[22] defined $T = t_1, t_2, \ldots, t_N$ as the recognition targets. For image classification, the classifier only needs one target – entire image ($N = 1$); For semantic segmentation, targets consist of all pixels ($N = \#ofpixels$); For object detection, targets consist of all proposals ($N = (\#ofpixels)^2$). Then the loss function sums up the loss from all targets. Instead of optimizing the loss from all targets, the authors performed an iterative optimization and only updated the loss for the targets correctly predicted in the previous iteration. The final perturbation sums up normalized perturbations in all iterations. To deal with a large number of targets for objective detection problem, the authors used regional proposal network (RPN) [4] to generate possible targets, which greatly decreases the computation for targets in object detection. DAG also showed the capability of generating images which are unrecognizable to human but deep learning could predict (*false positives*).

E. Semantic Segmentation

Image segmentation task can be viewed as an image classification task for every pixel. Since each perturbation is responsible for at least one pixel segmentation, this makes the space of perturbations for segmentation much smaller than that for image classification [120]. [22], [106], [120] generated adversarial examples against the semantic image segmentation task. However, their attacks are proposed under different scenarios. As we just discussed, [22] performed a non-targeted segmentation. [106], [120] both performed a targeted segmentation and tried to removed a certain class by making deep learning model to misguide it as background classes.

[106] generated adversarial examples by assigning pixels with the adversarial class that their nearest neighbor belongs to. The success rate was measured by the percentage of pixels of chosen class to be changed or of rest classes to be preserved.

[120] presented a method to generate universal adversarial perturbations against semantic image segmentation task. They assigned the primary objective of adversarial examples and hid the objects (*e.g.*, pedestrians) while keeping the rest segmentation unchanged. Metzen *et al.* defined background classes and targeted classes (not targeted adversarial classes). Targeted classes are the classes to be removed. Similar to [106], the pixels which belong to the targeted classes would be assigned to their nearest background classes:

$$\begin{aligned} l_{ij}^{target} &= l_{i'j'}^{pred} \quad \forall (i,j) \in I_{targeted}, \\ l_{ij}^{target} &= l_{ij}^{pred} \quad \forall (i,j) \in I_{background}, \\ (i',j') &= \underset{(i',j') \in I_{background}}{\arg\min} \quad \|i'-i\| + \|j'-j\|, \end{aligned} \tag{34}$$

where $I_{targeted} = (i,j) | f(x_{ij}) = l^*$ denotes the area to be removed. Figure 7 illustrates an adversarial example to hide

[10] This method was shown not effective for standard detectors (YOLO and Faster RCNN) later [119].

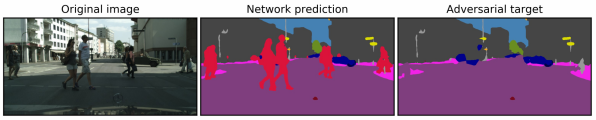

Figure 7: Adversary examples to hide pedestrians for semantic segmentation task [120]. Left image: original image; Middle image: the segmentation of the original image predicted by DNN; Right image: the segmentation of the adversarial image predicted by DNN.

pedestrians. They used *ILLC* attack to solve this problem and also extended *Universal Perturbation* method to get the universal perturbation. Their results showed the existence of universal perturbation for semantic segmentation task.

F. Natural Language Processing (NLP)

Many tasks in natural language processing can be attacked by adversarial examples. People usually generate adversarial examples by adding/deleting words in the sentences.

The task of reading comprehension (a.k.a. question answering) is to read paragraphs and answer questions about the paragraphs. To generate adversarial examples that are consistent with the correct answer and do not confuse human, Jia and Liang added distracting (adversarial) sentences to the end of paragraph [108]. They found that models for the reading comprehension task are overstable instead of oversensitivity, which means deep learning models cannot tell the subtle but critical difference in the paragraphs.

Hence, they proposed two kinds of methods to generate adversarial examples: 1) adding grammatical sentences similar to the question but not contradictory to the correct answer (AddSent); 2) adding a sentence with arbitrary English words (AddAny). [108] successfully fooled all the models (sixteen models) they tested on Stanford Question Answering Dataset (SQuAD) [121]. The adversarial examples also have the capability of transferability and cannot be improved by adversarial training. However, the adversarial sentences require manpower to fix the errors in the sentences.

[109] aimed to fool a deep learning-based sentiment classifier by removing the minimum subset of words in the given text. Reinforcement learning was used to find an approximate subset, where the reward function was proposed as $\frac{1}{\|D\|}$ when the sentiment label changes, and 0 otherwise. $\|D\|$ denotes the number of removing word set D. The reward function also included a regularizer to make sentence contiguous.

However, the changes in [108], [109] can easily be recognized by humans. More natural adversarial examples for texture data was proposed later [79] (Section IV-M).

G. Malware Detection

Deep learning has been used in static and behavioral-based malware detection because it can be well-generalized to zero-day malware [14]–[17]. [110]–[112], [114] generated adversarial malware samples to evade deep learning-based malware detection. Although all approaches were proposed against conventional machine learning classifiers, they used deep neural networks in their attacks. We introduce these attacks in this section.

Table IV: Summary of Countermeasures for Adversarial Examples

	Defensive Strategies	Representative Studies
Reactive	Adversarial Detecting	[34], [107], [122]–[129]
	Input Reconstruction	[127], [130], [131]
	Network Verification	[132]–[134]
Proactive	Network Distillation	[135]
	Adversarial (Re)Training	[35], [36], [55], [92], [94], [136]
	Classifier Robustifying	[137], [138]

[110] adapted *JSMA* method to attack Android malware detection model. [114] evaded two PDF malware classifier, PDFrate and Hidost, by modifying PDF. [114] parsed the PDF file and changed its object structure using genetic programming. The adversarial PDF file was then packed with new objects.

[113] used GAN to generate adversarial domain names to evade detection of domain generation algorithms. [112] proposed a GAN based algorithm, MalGan, to generate malware examples and evade black-box detection. [112] used a substitute detector to simulate the real detector and leveraged the transferability of adversarial examples to attack the real detector. MalGan was evaluated by 180K programs with API features. However, [112] required the knowledge of features used in the model. [111] used a large number of features (2350) to cover the required feature space of portable executable (PE) files. The features included PE header metadata, section metadata, import&export table metadata, *etc*. [111] also defined several modifications to generate malware evading deep learning detection. The solution was trained by reinforcement learning, where the reward function was the evasion rate.

VI. Countermeasures for Adversarial Examples

Countermeasures for adversarial examples have two main types of defense strategies: 1) *reactive*: detect adversarial examples after deep neural networks are built; 2) *proactive*: make deep neural networks more robust before adversaries generate adversarial examples. In this section, we discuss three kinds of reactive countermeasures (*Adversarial Detecting*, *Input Reconstruction*, and *Network Verification*) and three kinds of proactive countermeasures (*Network Distillation*, *Adversarial (Re)training*, and *Classifier Robustifying*). We will also discuss an ensembling methods to prevent adversarial examples. Table IV summarizes the countermeasures.

A. Network Distillation

Papernot *et al*. used network distillation to defend deep neural networks against adversarial examples [135]. Network distillation was originally designed to reduce the size of deep neural networks by transferring knowledge from a large network to a small one [139], [140] (Figure 8). The probability of classes produced by the first DNN is used as inputs to train the second DNN. The probability of classes extracts the knowledge learned from the first DNN. Softmax is usually used to normalize the last layer of DNN and produce the

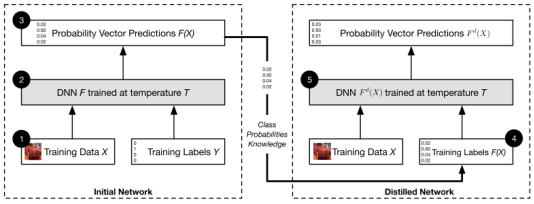

Figure 8: Network distillation of deep neural networks [135]

probability of classes. The softmax output of the first DNN, also the input of the next DNN, can be described as:

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}, \tag{35}$$

where T is a *temperature* parameter to control the level of knowledge distillation. In deep neural networks, *temperature* T is set to 1. When T is large, the output of softmax will be vague (when $T \to \infty$, the probability of all classes $\to \frac{1}{m}$). When T is small, only one class is close to 1 while the rest close to 0. The schema of network distillation can be repeated several times and connects several deep neural networks.

In [135], network distillation extracted knowledge from deep neural networks to improve robustness. The authors found that attacks primarily targeted the sensitivity of networks and then proved that using high-temperature softmax reduced the model sensitivity to small perturbations. "Network Distillation" was tested on MNIST and CIFAR10 and reduced the success rate of *JSMA* attack by 0.5% and 5% respectively. "Network Distillation" also improved the generalization of the neural networks.

B. Adversarial (Re)training

Training with adversarial examples is one of the countermeasures to make neural networks more robust. Goodfellow *et al*. [55] and Huang *et al*. [136] first included adversarial examples in the training stage. They generated adversarial examples in every step of training and inject them into the training set. [55], [136] showed that adversarial training improved the robustness of deep neural networks. Adversarial training could provide regularization for deep neural networks [55] and improve the precision as well [35].

[55] and [136] were evaluated only on the MNIST dataset. A comprehensive analysis of adversarial training methods on the ImageNet dataset was presented in [92]. They used half adversarial examples and half origin examples in each step of training. From the results, *adversarial training* increased the robustness of neural networks for *one-step attack* (*e.g*., *FGSM*) but would not help under iterative attacks (*e.g*., *BIM* and *ILLC* methods). [92] also suggested that adversarial training is used for regularization only to avoid overfitting (*e.g*., the case in [55] with the small MNIST dataset).

[94] found that the adversarial trained models on MNIST and ImageNet are more robust to white-box adversarial examples than to the transferred examples (black-box).

[36] minimized both the cross-entropy loss and internal representation distance during adversarial training, which can be seen as a defense version of *Feature Adversary*.

To deal with the transferred black-box model, [94] proposed *Ensembling Adversarial Training* method that trained the model with adversarial examples generated from multiple sources: the models being trained and also pre-trained external models.

C. Adversarial Detecting

A lot of research projects tried to detect adversarial examples in the testing stage [34], [107], [122]–[127], [129].

A few studies trained deep neural network-based binary classifiers as detectors to classify the input data as a legitimate (clean) input or an adversarial example [34], [107], [124]. Metzen *et al*. created a detector for adversarial examples as an auxiliary network of the original neural network [107]. The detector is a simple and small neural network predicting on binary classification, *i.e*., the probability of the input being adversarial. SafetyNet [34] extract the binary threshold of each ReLU layer's output as the features of the adversarial detector and detects adversarial images by RBF-SVM. The authors claimed that their method is hard to be defeated by adversaries even when adversaries know the detector, since it is difficult for adversaries to find an optimal value, for both adversarial examples and new features of SafetyNet detector. [125] added an outlier class to the original deep learning model. The model detects the adversarial examples by classifying it as an outlier. They found that the measurement of maximum mean discrepancy (MMD) and energy distance (ED) could successfully distinguish the distribution of adversarial datasets and clean datasets.

[123] provided a Bayesian view for detecting adversarial examples. [123] claimed that the uncertainty of adversarial examples is higher than the clean data. Hence, they deployed Bayesian neural networks to estimate the uncertainty of input data and used the uncertainty estimation to distinguish adversarial examples and clean input data.

Similarly, [127] used probability divergence (Jensen-Shannon divergence) as one of its detectors. [126] showed that after whitening by Principal Component Analysis (PCA), adversarial examples have different coefficients in low-ranked components.

[131] trained a PixelCNN neural network [141] and found that the distribution of adversarial examples is different from clean data. They calculated p-value based on the rank of PixelCNN and rejected adversarial examples using the p-values. The results showed that this approach could detect *FGSM*, *BIM*, *DeepFool*, and *C&W* attack.

[128] first trained neural networks with "reverse cross-entropy" to better distinguish adversarial examples from clean data in the latent layers and then detected adversarial examples using a method called "Kernel density" in the testing stage. The "reverse cross-entropy" made the deep neural network to predict a high confidence on the true class and a uniform distribution on the other classes. In this way, the deep neural network was trained to map the clean input close to a low-dimensional manifold in the layer before softmax. This brought great convenience for further detection of adversarial examples.

[129] leveraged multiple previous images to predict future input and detect adversarial examples, in the task of reinforcement learning.

However, Carlini and Wagner summarized most of these adversarial detecting methods ([107], [122]–[126]) and showed that these methods could not defend against their previous attack *C&W's Attack* with slight changes of loss function [96], [97].

D. Input Reconstruction

Adversarial examples can be transformed to clean data via reconstruction. After transformation, the adversarial examples will not affect the prediction of deep learning models. Gu and Rigazio proposed a variant of denoising autoencoder network with a penalty, called deep contractive autoencoder, to increase the robustness of neural networks [130]. The autoencoder network is trained from adversarial examples to original ones and from original samples to themselves. [127] reconstructed the adversarial examples by 1) adding Gaussian noise or 2) encoding them with autoencoder as a plan B in MagNet [127](Section VI-G).

PixelDefend reconstructed the adversarial images back to the training distribution [131] using PixelCNN. *PixelDefend* changed all pixels along each channel to maximize the probability distribution:

$$\begin{aligned} \max_{x'} \quad & \mathcal{P}_t(x') \\ s.t. \quad & \|x' - x\|_\infty \leq \epsilon_{defend}, \end{aligned} \tag{36}$$

where $\mathcal{P}_t$ denotes the training distribution, ϵ_{defend} controls the new changes on the adversarial examples. *PixelDefend* also leveraged adversarial detecting, so that if an adversarial example is not detected as malicious, no change will be made to the adversarial examples (set ϵ_{defend} as 0).

E. Classifier Robustifying

A robust architecture of deep neural networks can prevent adversarial examples.

Due to the uncertainty from adversarial examples, Bradshaw *et al*. leveraged Bayesian classifiers to build more robust neural networks [137]. Gaussian processes (GPs) with RBF kernels were used to provide uncertainty estimation. The proposed neural networks were called *Gaussian process hybrid deep neural networks (GPDNNs)*. GPs expressed the latent variables as a Gaussian distribution parameterized by the functions of mean and covariance and encoded them with RBF kernels. [137] showed that GPDNNs achieved comparable performance with general DNNs and more robust to adversarial examples. The authors claimed that GPDNNs "know when they don't know."

[138] observed that adversarial examples usually went into a small subset of incorrect classes. [138] separated the classes into sub-classes and ensembled the result from all sub-classes by voting to prevent adversarial examples misclassified.

F. Network Verification

Verifying properties of deep neural networks is a promising solution to defend adversarial examples, because it may prevent the new unseen attacks. *Network verification* checks the properties of a neural network: whether an input violates or satisfies the property.

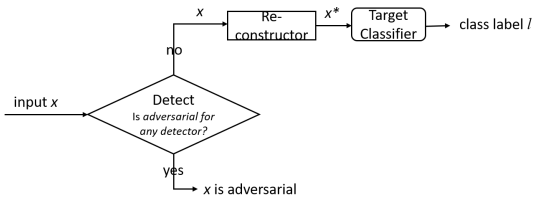

Figure 9: MagNet workflow: one or more detectors first detects if input x is adversarial; If not, reconstruct x to x^* before feeding it to the classifier. (modified from [127])

Katz *et al*. proposed a verification method for neural networks with ReLU activation function, called Reluplex [132]. They used Satisfiability Modulo Theory (SMT) solver to verify the neural networks. The authors showed that within a small perturbation, there was no existing adversarial example to misclassify the neural networks. They also proved that the problem of network verification is NP-complete. Carlini *et al*. extended their assumption of ReLU function by presenting $\max(x, y) = ReLU(x-y)+y$ and $\|x\| = ReLU(2x)-x$ [90]. However, Reluplex runs very slow due to the large computation of verifying the networks and only works for DNNs with several hundred nodes [134]. [134] proposed two potential solutions: 1) prioritizing the order of checking nodes 2) sharing information of verification.

Instead of checking each point individually, Gopinath *et al*. proposed *DeepSafe* to provide safe regions of a deep neural network [133] using Reluplex. They also introduced *targeted robustness* a safe region only regarding to a targeted class.

G. Ensembling Defenses

Due to the multi-facet of adversarial examples, multiple defense strategies can be performed together (parallel or sequential) to defend adversarial examples.

Aforementioned *PixelDefend* is composed of an adversarial detector and an "input reconstructor" to establish a defense strategy.

MagNet included one or more detectors and a reconstructor (referred as "reformer" in the paper) as Plan A and Plan B [127]. The detectors are used to find the adversarial examples which are far from the boundary of the manifold. In [127], they first measured the distance between input and encoded input and also the probability divergence (Jensen-Shannon divergence) between softmax output of input and encoded input. The adversarial examples were expected a large distance and probability divergence. To deal with the adversarial examples close to the boundary, MagNet used a reconstructor built by neural network based autoencoders. The reconstructor will map adversarial examples to legitimate examples. Figure 9 illustrates the workflow of the defense of two phases.

After investigating several weak defenses, [142] showed that the ensemble of weak defensive approaches do not make the neural networks strong.

H. Summary

Almost all defenses are shown to be effective only for the weak attacks. They tend not to be defensive for strong and unseen attacks. Most of defenses target adversarial examples in the computer vision task. However, with the development of adversarial examples in other areas, new defenses for these areas, especially for safety-critical systems, are urgently required.

VII. Challenges and Discussions

In this section, we discuss the current challenges and the potential solutions for adversarial examples. Although many methods and theorems have been proposed and developed in recent years, a lot of fundamental questions need to be well explained, and a lot of challenges need to be addressed. The reason for the existence of adversarial examples is an interesting and one of the most fundamental problems for both adversaries and researchers, which exploits the vulnerability of neural networks and help defenders to resist adversarial examples. We will discuss the following questions in this section: Why do adversarial examples transfer? How to stop the transferability? Why are some defenses effective and others not? How to measure the strength of an attack as well as a defense? How to evaluate the robustness of a deep neural network against seen/unseen adversarial examples?

A. Transferability

Transferability is a common property for adversarial examples. Szegedy *et al*. first found that adversarial examples generated based on a neural network can fool the same neural networks trained by different datasets. Papernot *et al*. found that adversarial examples generated based on a neural network can fool other neural networks with different architectures, even other classifiers trained by different machine learning algorithms [44]. Transferability is critical for black-box attacks where the victim deep learning model and the training dataset are not accessible. Attackers can train a substitute neural network model and then generate adversarial examples against substitute model. Then victim will be vulnerable to these adversarial examples due to transferability. From a defensive view, if we stop transferability of adversarial examples, we can defend all white-box attackers who need to access the model.

We define the transferability of adversarial examples in three dimensions from easy to hard: 1) transfer among the same neural network architecture trained with different data; 2) transfer among different neural network architectures trained for the same task; 3) transfer among deep neural networks for different tasks. To our best knowledge, there is no existing solution on the third dimension yet (for instance, transfer an adversarial image from object detection to semantic segmentation).

Many studies examined transferability to show the ability of adversarial examples [19], [55]. Papernot *et al*. studied the transferability between conventional machine learning techniques (*i.e*., logistic regression, SVM, decision tree, kNN) and deep neural networks. They found that adversarial examples can be transferred between different parameters, training dataset of a machine learning models and even across different machine learning techniques.

Liu *et al*. investigated transferability of targeted and non-targeted adversarial examples with large models and large datasets (*e.g*., the ImageNet dataset) [89]. They found that non-targeted adversarial examples are much more transferable than targeted ones. They observed that the decision boundary of non-targeted adversarial aligns well with each other. Thus they proposed *Model-Based Ensembling Attack* to create transferable targeted adversarial examples.

Tramèr *et al*. found that the distance to the model's decision boundary is on average larger than the distance between two models' boundaries in the same direction [143]. This may explain the existence of transferability of adversarial examples. Tramèr *et al*. also claimed that transferability might not be an inherent property of deep neural networks by showing a counter-example.

B. Existence of Adversarial Examples

The reason for the existence of adversarial examples is still an open question. Are adversarial examples an inherent property of deep neural networks? Are adversarial examples the "Achilles' heel" of deep neural networks with high performance?

Many papers have proposed hypotheses to explain the existence. [19] suggested that adversarial examples is of low probability, or even never observed data samples in the testing dataset. Deep neural networks are fooled due to covariate shift. From training a PixelCNN, [131] found that the distribution of adversarial examples was different from clean data. [55] claimed that adversarial examples occurred in a large and contiguous space instead of randomly scattered. From the experiments in [143], adversarial examples transferring between two small neural networks formed a 25-dimensional space.

[55] suggested that adversarial examples are the results of models being too linear in high dimensional manifolds. [144] showed that in the linear case, the adversarial examples exist when the decision boundary is close to the manifold of the training data.

Contrary to [55], [145] believed that adversarial examples are due to the "low flexibility" of the classifier for certain tasks. [88] also found that linearity is not an "obvious explanation". [44], [145] showed that adversarial examples are a phenomenon not only for deep neural networks but also for all classifiers.

[91] blamed adversarial examples for the sparse and discontinuous manifold which makes classifier erratic. [36] suggested that the decision boundaries of deep neural networks are inherently incorrect, which do not detect semantic objects.

Instead of analyzing how to train a robust deep neural network, [146] claimed that adversarial examples are due to low test coverage of corner cases.

Besides adversarial examples for image classification task, as shown in Section V, adversarial examples have been generated in various applications and scenarios. Many of them deployed completely different methods. Some applications can use the same method used in image classification task.

However, some need to propose a novel method. Current studies on adversarial examples mainly focuses on image classification task. No existing paper explains the relationship among different applications. Does it exist a universal attacking/defending method to be applied to all the applications?

C. Robustness Evaluation

The competition between attacks and defenses for adversarial examples becomes an "arms race": a defensive method that was proposed to prevent existing attacks was later shown to be vulnerable to some new attacks, and vice versa [90], [125]. Some defenses showed that they could defend a certain attack, but later failed with a slight change of the attack [122], [123]. Hence, the evaluation on the robustness of a deep neural network is necessary. [147] provided an upper bound of robustness for linear classifier and quadratic classifier. The following problems for robustness evaluation on deep neural networks require further exploration.

1) **A methodology for evaluation on the robustness of deep neural networks**: Many deep neural networks are planned to be deployed in safety-critical settings. Defending only existing attacks is not sufficient. Zero-day (new) attacks would be more harmful to deep neural networks. A methodology for evaluating the robustness of deep neural networks is required, especially for zero-day attacks, which helps people understand the confidence of model prediction and how much we can rely on them in the real world. [90], [132], [148], [149] conducted initial studies on the evaluation. Moreover, this problem lies not only in the performance of deep neural network models but also in the confidentiality and privacy.

2) **A benchmark platform for attacks and defenses**: Most attacks and defenses described their methods without publicly available code, not to mention the parameters used in their methods. This brings difficulties for other researchers to reproduce their solutions and provide the corresponding attacks/defenses. For example, Carlini tried his best to "find the best possible defense parameters + random initialization"[11]. Some researchers even drew different conclusions because of different settings in their experiments. If there exists any benchmark, where both adversaries and defenders conduct experiments in a uniform way (such as same scenarios, datasets, classifiers, used attacking/defending techniques), we can make fairly clearer comparisons between different attacking and defending techniques. Cleverhans [150] and Foolbox [151] are open-source libraries to benchmark the vulnerability of deep neural networks against adversarial images. They build great frameworks to benchmark the attacks. However, defensive strategies are missing in both tools. Providing a dataset of adversarial examples generated by different methods will make it easy for finding the blind point of deep neural networks and developing new defense strategies. This problem also occurs in other areas in deep learning.

We present a workflow of a benchmark platform for attackers and defenders (Figure 10).

[11]Code repository used in [96]: https://github.com/carlini/nn_breaking_detection

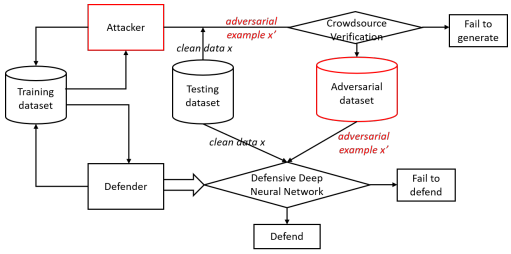

Figure 10: Workflow of a benchmark platform for attackers and defenders: 1) attackers and defenders update/train their strategies on training dataset; 2) attackers generate adversarial examples on the clean data; 3) the adversarial examples are verified by crowdsourcing whether recognizable to human; 4) defenders generate a deep neural network as a defensive strategy; 5) evaluate the defensive strategy.

3) **Various applications for robustness evaluation**: Similar to the existence of adversarial examples for various applications, a wide range of applications make it hard to evaluate the robustness, of a deep neural network architecture. How to compare methods generating adversarial example under different scenario? Do we have a universal methodology to evaluate the robustness under all scenarios? Tackling these unsolved problems is a future direction.

VIII. Conclusion

In this paper, we reviewed recent findings of adversarial examples in deep neural networks. We investigated existing methods for generating adversarial examples[12]. A taxonomy of adversarial examples was proposed. We also explored the applications and countermeasures for adversarial examples.

This paper attempted to cover state-of-the-art studies for adversarial examples in the deep learning domain. Compared with recent work on adversarial examples, we analyzed and discussed current challenges and potential solutions lying in adversarial examples.

Acknowledgment

The work presented is supported in part by National Science Foundation (grants ACI 1245880, ACI 1229576, CCF-1128805, CNS-1624782), and Florida Center for Cybersecurity seed grant.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[3] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.
[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.
[5] G. Saon, H.-K. J. Kuo, S. Rennie, and M. Picheny, "The ibm 2015 english conversational telephone speech recognition system," *arXiv preprint arXiv:1505.05899*, 2015.
[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[12]Due to the rapid development of adversarial examples (attacks and defenses), we only considered the papers published before November 2017. We will update the survey with new methodologies and papers in our future work.