Vulnerability Detection and Analysis in Adversarial Deep Learning

Yi Shi, Yalin E. Sagduyu, Kemal Davaslioglu and Renato Levy

Abstract Machine learning has been applied in various information systems, but its vulnerability has not been well understood yet. This chapter studies vulnerability to *adversarial machine learning* in information systems such as online services with interfaces that accept user data inputs and return machine learning results such as labels. Two types of attacks are considered: *exploratory (or inference) attack* and *evasion attack*. In an *exploratory attack*, the adversary collects labels of input data from an online classifier and applies *deep learning* to train a functionally equivalent classifier without knowing the inner working of the target classifier. The vulnerability includes the theft of intellectual property (quantified by the statistical similarity of the target and inferred classifiers) and the support of other attacks built upon the inference results. An example of follow-up attacks is the *evasion attack*, where the adversary deceives the classifier into misclassifying input data samples that are systematically selected based on the classification scores from the inferred classier. This attack is strengthened by *generative adversarial networks* (GANs) and *adversarial perturbations* producing *synthetic data* samples that are likely to be misclassified. The vulnerability is measured by the increase in misdetection rates. This quantitative understanding of the vulnerability in machine learning systems provides valuable insights into designing defence mechanisms against adversarial machine learning.

Y. Shi (✉) · Y. E. Sagduyu · K. Davaslioglu · R. Levy
Intelligent Automation, Inc., 15400 Calhoun Drive, Rockville, MD 20855, USA
e-mail: yshi@i-a-i.com

Y. E. Sagduyu
e-mail: ysagduyu@i-a-i.com

K. Davaslioglu
e-mail: kdavaslioglu@i-a-i.com

R. Levy
e-mail: rlevy@i-a-i.com

S. Parkinson et al. (eds.), *Guide to Vulnerability Analysis for Computer Networks and Systems*, Computer Communications and Networks,
https://doi.org/10.1007/978-3-319-92624-7_9

1 Introduction

Information systems are susceptible to various exploits and attacks that range from passing phishing emails through spam filters to stealing intellectual property. These threats apply not only to traditional programs or services where all behaviours are programmed manually, but also to *machine learning*-based systems that are becoming increasingly prevalent in current information systems. For example, cybersecurity systems started applying machine learning to analyse and detect malicious traffic components as automated means for intrusion detection and prevention [1].

Facilitated by the availability of fast hardware and large datasets, *deep learning* has emerged as a powerful machine learning approach that can solve complex problems such as defeating the best players in complex games; e.g. Google's AlphaGo [2] defeated a Go master. Deep learning involves training *deep neural network* structures to learn to approximate complex functions that map input data to outputs (e.g. assign labels to data samples in classification problems). As information systems start relying more on machine learning, adversaries turn their attention to directly attack the underlying machine learning functionalities that involve sensitive and proprietary information, e.g. training data, machine learning algorithm and its hyperparameters, and functionality of underlying tasks. One seminal example in image recognition systems is that adversarial images generated by slightly perturbing real images via deep learning have fooled a state-of-the-art image classifier into recognizing a perturbed panda image as a gibbon [3].

Attacks on machine learning, often referred to as *adversarial machine learning* [4–6], have raised the need to understand how effective machine learning can take place under the presence of an adversary. One particular application of adversarial machine learning is tailored to online information systems, in which the adversaries interact with systems through application programming interfaces (APIs). Typically offered online, software as a service (SaaS) or software on demand has become a popular software distribution model that hosts applications and makes them available to customers over the Internet. Without revealing source codes or inner workings of the underlying algorithms, these services are usually provided via subscriptions, while still offering free public access either in trial periods or subject to rate limitations on the frequency and volume of access to services. This paradigm allows adversaries to seek the opportunity to attack machine learning systems. Clear understanding of vulnerabilities to these attacks is of paramount importance to secure deployment of machine learning systems.

While the field of adversarial machine learning is growing, the security limitations of machine learning ranging from *intellectual property theft* to *system malfunction* are not well understood yet and deserve further study to identify effects of adversarial machine learning and provide the foundation for appropriate attack detection and mitigation techniques. To improve system security, it is critical to quantify the *vulnerability* of machine learning to various exploits and attacks, and assess the impact of this vulnerability on tasks performed by machine learning.

This chapter studies detecting and analysing vulnerabilities of information systems to adversarial machine learning. Deep learning is the focus of this study to understand the practical limits on adversaries when they attack machine learning-based systems. An image classification service is considered, while the design guidelines are generic and readily applicable to other types of data such as text. Two types of attacks based on adversarial deep learning are considered. The first one is the *exploratory attack*, where the adversary aims to infer information on the machine learning system under attack without knowledge of the original classifier's algorithm, parameters and training datasets [7]. A *deep neural network* is trained by the adversary for that purpose. The second one is the *evasion attack*, where the adversary aims to fool the machine learning systems into making wrong decisions [8]. The adversary first runs data samples through the inferred classifier and then sends to the target classifier only those samples that are assigned classification scores close to the decision boundary. Two approaches of *generative adversarial networks* (GANs) [9] and *adversarial perturbations* [3] based on deep learning are used to strengthen the evasion attack by generating additional synthetic data samples that are likely to be misclassified.

The potential success of these attacks is quantified as indicators of vulnerabilities in the underlying machine learning system. The vulnerability to the exploratory attack is quantified as the statistical similarity of the output labels returned by the target classifier and inferred classifier. The vulnerability to the evasion attack is quantified as the misdetection error incurred by the target classifier with respect to real and synthetic adversarial inputs. Based on the insights gained through vulnerability analysis, defence mechanisms can be developed against adversarial deep learning. The impact of adversarial deep learning is diverse and far-reaching beyond cybersecurity applications. Various other domains such as wireless or mobile security benefit from the insights gained through the analysis of vulnerability to adversarial deep learning in information systems.

The rest of this chapter is organized as follows. Section 2 presents basic principles of deep learning. Section 3 introduces the problem of adversarial machine learning. Section 4 analyses the vulnerability of machine learning to exploratory attacks. Section 5 analyses the vulnerability of machine learning to evasion attacks. Section 6 analyses the impact of adversarial synthetic inputs in evasion attacks. Section 7 describes insights on defence mechanisms based on vulnerability analysis. Section 8 discusses extension of adversarial machine learning to other domains. Section 9 presents end-of-chapter teaching questions. Section 10 presents end-of-chapter research questions.

2 Deep Learning

Deep learning refers to training a deep neural network. Deep learning can be applied to many applications ranging from detection to prediction. In this chapter, we consider classification tasks that are trained to assign a label or a class to each data sample.

A sample s can be described by a set of M features, $F_s = \{f_s^m\}_{m=1}^M$, and belongs to a class $C(s) \in \mathscr{C}$. A classifier T will classify a given sample s as the class $T(s) \in \mathscr{C}$.

Ideally, a classifier should provide the correct class of any given sample s, i.e. $T(s) = C(s)$. However, even with sophisticated machine learning algorithms and large training datasets, there might be still samples with $T(s) \neq C(s)$; i.e. they might be misclassified. We use error probability to measure the accuracy of a classifier. In particular, we measure the following errors for a given set of test data with respect to the ground truth:

1. Error probability on class i is given by

$$e_i(T) = \frac{\sum_{j \in \mathscr{C}}^{j \neq i} e_{ij}(T)}{n_i}, \tag{1}$$

   where $e_{ij}(T)$ is the number of samples that are in class i but are classified as class j (i.e. $C(s) = i$ and $T(s) = j$) and n_i is the total number of samples in class i (i.e. $C(s) = i$).
2. Overall error probability is given by

$$e(T) = \frac{\sum_{i \in \mathscr{C}} \sum_{j \in \mathscr{C}}^{j \neq i} e_{ij}(T)}{\sum_{i \in \mathscr{C}} n_i}. \tag{2}$$

In addition to deep learning, there are many other types of machine learning algorithms, such as Naive Bayes and Support Vector Machine (SVM), that can also be applied for the classification problem. We will apply them at the classifier under the attack.

1. Naive Bayes is a probabilistic classifier based on applying Bayes' theorem with (naive) independence assumptions between the features. Due to this assumption, it is important to select features that they are (almost) independent. Note that such a requirement does not exist for deep learning, which makes feature selection easier. Naive Bayes builds conditional probabilities of features (represented as vectors of feature values) under each class and determines the most likely class for each sample.
2. SVM is a non-probabilistic classifier that represents samples in training data as points in a high-dimensional feature space and assumes that points in different classes can be separated by hyperplanes. Each hyperplane is optimized to maximize the gap between the hyperplane and points in different classes separated by this hyperplane. SVM maps samples in test data into the same space and predicts their classes based on the optimal hyperplanes. Note that the hyperplane assumption does not exist for deep learning and thus it can use any surfaces to separate points in different classes.

Table 1 shows the notation used in this chapter. In the subsequent section, we will present different deep learning techniques for the classification problem.

Table 1 Notation

$\hat{\mathscr{A}}_{ij}$	The attack region to misclassify class i samples as in class j
b_j	A constant bias term used by neuron j
$\mathbf{b}_l$	The vector of biases at the lth layer
$C(s)$	The class of sample s
$\mathscr{C}$	The set of all classes
$d_{ij}(T)$	The number of samples that have different classes by T and $\hat{T}$
$d_i(T, \hat{T})$	The difference between T and $\hat{T}$ on class i
$d(T, \hat{T})$	The difference between T and $\hat{T}$
$d(\underline{r}, \hat{\mathscr{R}})$	The distance between a point $\underline{r}$ and a region $\hat{\mathscr{R}}$
$d(\underline{r}, \underline{x})$	The distance between two points $\underline{r}$ and $\underline{x}$
D	The discriminator network in GAN
$e_{ij}(T)$	The number of samples that are in class i but are classified as class j by T
$e_i(T)$	T's error probability on class i
$e(T)$	T's overall error probability
$f(\mathbf{x}_0; \theta)$	The mapping function of FNN
f_s^m	The mth feature of sample s
F_s	$= \{f_s^m\}$, a set of features of sample s
G	The generator network in GAN
$G(z)$	The generator output for z
$J(\theta, s)$	The cost used to train the neural network
M	The number of samples in evasion attack
n	The number of hidden layers
n_i	The number of samples in class i
$\hat{n}_i$	The number of samples classified by T as in class i
N	The number of queries for exploratory attack
N_{class}	The number of classes
$\mathbb{P}_{data}$	The distribution of input data r
$\mathbb{P}_z$	The model distribution of noise z
$\underline{r}(s)$	A vector of T's scores for sample s
$\hat{\underline{r}}(s)$	A vector of $\hat{T}$'s scores for sample s
$r_i(s)$	The score for sample s being in class i
$\mathscr{R}_i$	The region of T's scores for samples in class i
$\hat{\mathscr{R}}_i$	The region of $\hat{T}$'s scores for samples in class i
s	A sample
T	The target classifier under attack
$\hat{T}$	The inferred classifier
$T(s)$	The identified class for sample s by T
w_{ji}	The weight from neuron i to neuron j
W_l	The matrix of weights at the lth layer
$\mathbf{x}_l$	The output of lth layer
y_j	The output of neuron j

(continued)

Table 1 (continued)

z	A noise input to generator G
$\sigma(x)$	A nonlinear activation function used in neural networks
θ_l	$= \{W_l, \mathbf{b}_l\}$, weight and biases at the lth layer
θ	$= \{\theta_1, \ldots, \theta_L\}$, the set of all parameters of FNN
$\hat{\theta}$	The threshold for a binary classification problem
ϕ	The paragraph to define an attack region
$\Delta(s)$	The minimum perturbation to change the class predicted by T for a sample s

2.1 Artificial Neural Networks

An *artificial neural network* is a machine for computing some function, and consists of simple elements called neurons that are joined by weighted connections also known as synapses. A neuron j performs a basic computation over its input synapses with weight w_{ji} (connecting neurons i and j) and a bias term b_j, and outputs a single scalar value y_j, which can be interpreted as its activation, or a firing rate. For example, the following computation is commonly used:

$$y_j = \sigma\left(b_j + \sum_{i \neq j} w_{ji} y_i\right), \tag{3}$$

where $\sigma(x)$ is some nonlinear activation function such as the sigmoid function

$$\sigma(x) = 1/\left(1 + e^{-x}\right) . \tag{4}$$

Figure 1 shows the common feedforward neural network (FNN) architecture (topology), where circles and lines denote neurons and synapses, respectively, and the neurons are arranged in layers. The activations of neurons in the input layer (left) are set externally, while the activations of the hidden layer neurons (middle) and output layer neurons (right) are computed as specified above; the latter represent the result of the network's computation.

For the classification problem, the activations of neurons in the input layer are set by features of a sample, while the computed results at the output layer show the class for a sample. Provided that the hidden layer has a sufficiently large number of neurons, this simple architecture can (in theory) approximate any mathematical function to an arbitrary small error probability via its synaptic weights, which is called the *universal approximation theorem* [10, 11]. However, for many real-world problems, a good approximation would often require a prohibitively large number of hidden layer neurons.

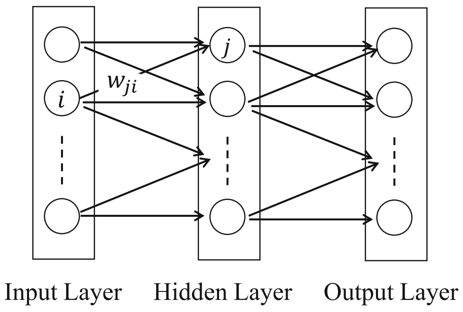

Fig. 1 Structure of an artificial neural network

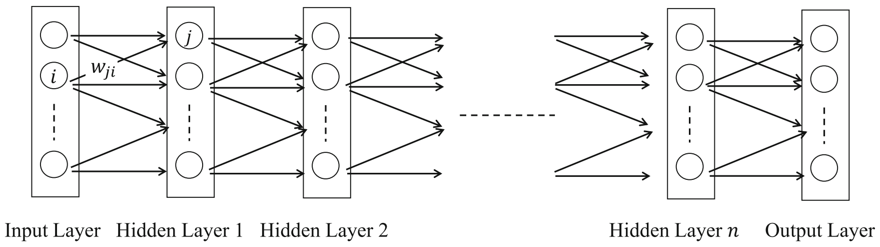

Fig. 2 Deep neural network

2.2 *Deep Neural Networks*

To overcome the challenge of a prohibitively large number of hidden layer neurons, neural network designs have focused on deep architectures that contain many hidden layers (as shown in Fig. 2). The training of such architectures is called *deep learning* and can be time-consuming, because learning is typically achieved through gradient descent on the network's output error (defined as a function of its weights), and the components of this gradient are often very small for the weights of connections that are far away from the input layer, which is known as the *vanishing gradient problem*. However, the process is now facilitated by the availability of fast hardware and large datasets. Once trained, the layers of the deep neural network can transform raw data into representations that become progressively more sophisticated and abstract with each layer, allowing the network to produce complex decisions (e.g. the class for a sample) at the output layer. As a result, deep learning has achieved unprecedented results in areas such as speech processing, machine translation and computer vision and is presently being utilized within (or integrated into) many practical applications and products.

There are various types of neural networks. Three main types are as follows:

1. *Feedforward neural network (FNN)*, also called multilayer perceptron (MLP), consists of n hidden layers and describes a mapping of an input vector through n iterative processing steps, as shown in Fig. 2. FNNs will be used primarily in this

chapter for adversarial deep learning and will be discussed in more detail in the next section to provide insights into training and testing in deep neural network structures.

2. *Convolutional neural network (CNN)* [12] connects a hidden neuron to only a subset of neurons in the previous layer such that multiple hidden neurons share the same weights, thus greatly reducing the space of parameters to be learned. CNNs can be effectively applied to machine learning problems with spatially correlated data. In the CNN architecture, the 2-D convolutional layer applies sliding filters to the input. The max pooling layer progressively reduces the spatial size of the representation and controls overfitting. The batch normalization layer normalizes each input channel across a mini-batch to speed up training and reduce the sensitivity to network initialization. CNNs will be used primarily in this chapter for synthetic data generation purposes.
3. *Recursive neutral network* allows sequence processing where the synaptic connections form cycles, and enables activation to persist between time steps. A limitation of most deep neural networks such as FNN and CNN is that they are memoryless; thus, every input is processed independently of any other inputs, and the network is not able to take advantage of temporal patterns within an input sequence, when making a decision. A particular type of RNN is the long short-term memory network (LSTM) [13], where memory neurons store information through self-activation and their activity is modulated by specialized gate neurons. The input gates determine the extent to which the current input should be stored in memory; the forget gates determine how much memory is retained between time steps, and finally, the output gate modulates the effect of memory upon the current output. Through its memory and gating mechanisms, LSTM is able to remember relevant information for arbitrary lengths of time and capture long-range patterns within data sequence.

2.3 Feedforward Neural Networks

An FNN learns to map an input vector $\mathbf{x}_0$ to the output vector $\mathbf{x}_n$ (namely, labels) by following n steps (one for each layer):

$$\mathbf{x}_l = f_l(\mathbf{x}_{l-1}; \theta_l), \quad l = 1, \ldots, n, \tag{5}$$

where the set of parameters (weights W_l and biases $\mathbf{b}_l$) in the lth layer is denoted by θ_l and the lth layer performs the mapping

$$f_l\left(\mathbf{x}_{l-1}; \theta_l\right) = \sigma\left(W_l \mathbf{x}_{l-1} + \mathbf{b}_l\right). \tag{6}$$

In this mapping, W_l denotes the matrix of weights, $\mathbf{b}_l$ denotes the vector of biases, and $\sigma(\cdot)$ denotes an activation function. Examples of activation functions are rectifying linear unit (ReLU) activation function that defines the mapping

$$\sigma(u_i) = \max(0, u_i) \tag{7}$$

and softmax activation function (a generalization of the logistic function) that defines the mapping

$$\sigma(u_i) = e^{u_i} / \sum_j e^{u_j}. \tag{8}$$

An FNN is trained using labelled training data, i.e. a set of input–output vector pairs $(\mathbf{x}_{0,i}, \mathbf{x}^*_{n,i})$, $i = 1, \ldots, S$, where $\mathbf{x}^*_{n,i}$ is the desired output of the neural network when $\mathbf{x}_{0,i}$ is used as input. The goal of training is to minimize a loss function such as mean squared error or cross-entropy error. The set of parameters θ are derived by applying stochastic gradient descent (SGD). The gradient is computed through the *backpropagation algorithm* that consists of two phases: propagation and weight update [14].

A deep learning classifier provides not only a decision on whether a sample s is in class $i \in \mathscr{C}$, i.e. $T(s) = i$, but also a set of scores $\underline{r}(s)$, where $r_i(s)$ is the score for sample s under class $i \in \mathscr{C}$. One example of score is the output of the softmax function at the output layer. After parameter optimization in the training phase to determine decision boundaries, the region of all $\underline{r}(s)$ values is divided into subregions $\mathscr{R}_i$ for $i \in \mathscr{C}$, such that if $\underline{r}(s) \in \mathscr{R}_i$, then the decision $T(s) = i$ is made. Numerical results in this chapter are based on the implementation of deep learning with the Microsoft Cognitive Toolkit (CNTK) [15] and TensorFlow [16].

3 Adversarial Deep Learning

A machine learning algorithm or an information system built upon one can be “attacked” in several ways [4–6, 17] (illustrated in Fig. 3):

1. *Exploratory* attacks, e.g. [18–20], occur after the algorithm has been trained, and attempt to uncover information about its inner workings, in order to identify vulnerability of the algorithm. This attack may attempt to extract several factors such as

   - the decision boundary used by the algorithm (e.g. hyperplanes of the Support Vector Machine (SVM) algorithm),
   - a general set of rules that the algorithm follows,
   - a set of logical or probabilistic properties about the algorithm, or
   - information about the data that was used (or not used) to train the algorithm.

2. *Evasion* attacks, e.g. [21–23], are also launched upon trained algorithms, and involve providing the algorithm with input (test) data that will result in an incorrect output.

   - A classic example of an evasion attack is the generation of spam emails that will fool a trained filter into accepting them as legitimate.

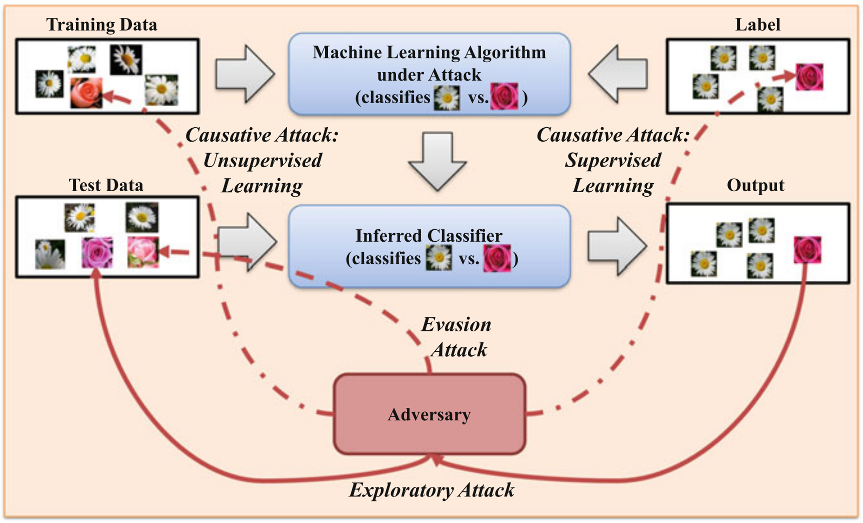
Training Data
Machine Learning Algorithm under Attack (classifies vs.)
Label
Causative Attack: Unsupervised Learning
Causative Attack: Supervised Learning
Test Data
Inferred Classifier (classifies vs.)
Output
Evasion Attack
Adversary
Exploratory Attack

Fig. 3 Adversarial machine learning

- Another example is the creation of social bots (e.g. automated tweet generators) that will fool bot detectors.
- Evasion attacks can be made more effective by exploiting vulnerabilities identified via exploratory attacks, or created via causative attacks.

3. *Causative* attacks (also known as poisoning attacks), e.g. [24, 25], attempt to provide incorrect training data to the algorithm, such that it does not learn the intended function.

    - In supervised learning, this may be achieved by mislabelling the training data, e.g., in adversarial active learning [5], where the algorithm requests training labels from potentially untrusted sources.
    - In reinforcement learning, a causative attack can take place by providing incorrect rewards or punishments.
    - In unsupervised learning, the attack can be launched by sampling training examples in a biased way, such that they do not reflect the underlying statistical distribution.

The chapter focuses on exploratory and evasion attacks that target the test phase (instead of training phase) of machine learning. In the next two sections, we will provide more discussion on exploratory and evasion attacks.

4 Exploratory Attack

Many systems provide online services that allow a user to provide input data (e.g. through an API or through a Web interface) and observe the output (e.g. labels as classification output). Such systems are vulnerable to *evasion*, *exploratory*, and in some cases *causative* attacks based on adversarial machine learning. In this section, we address exploratory attacks. The state of the art focuses on *model inversion attacks* [18–20], where the adversary typically knows the type and structure of the classifier, and attempts to learn the model parameters. In this chapter, we consider a generalized case for the adversary and let the adversary launch an attack to an online classification service by building *a functionally equivalent classifier without knowing the type, structure and parameters of the classifier*. This is called *black-box attack* [7, 26].

As shown in Fig. 4, the adversary launches a three-step attack to build a functionally equivalent classifier:

1. The adversary polls the target classifier with input data.
2. The adversary observes labels returned by the target classifier.

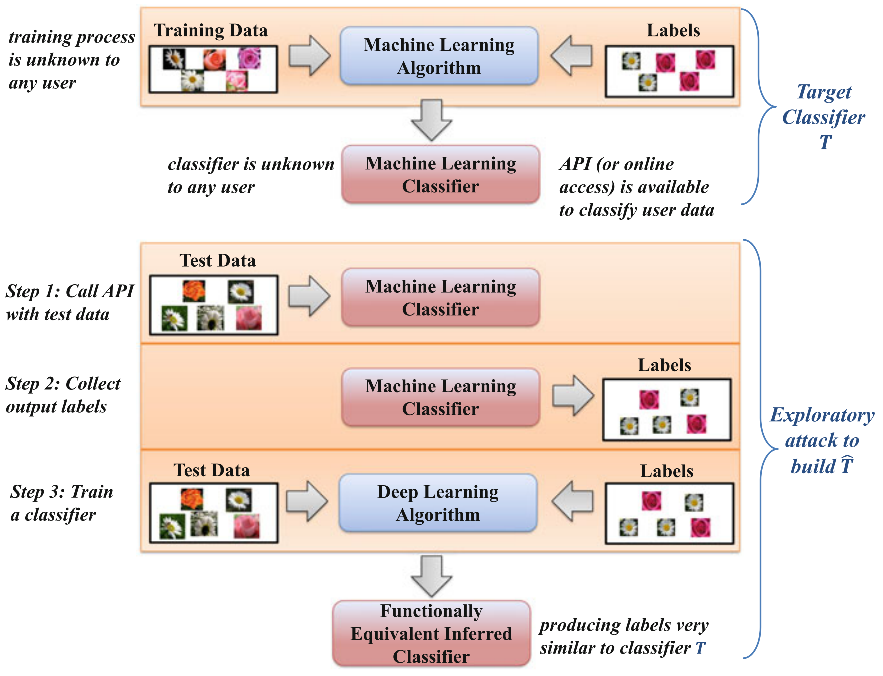
training process is unknown to any user
Training Data
Machine Learning Algorithm
Labels
Target Classifier T
classifier is unknown to any user
Machine Learning Classifier
API (or online access) is available to classify user data
Step 1: Call API with test data
Test Data
Machine Learning Classifier
Step 2: Collect output labels
Machine Learning Classifier
Labels
Exploratory attack to build $\hat{T}$
Step 3: Train a classifier
Test Data
Deep Learning Algorithm
Labels
Functionally Equivalent Inferred Classifier
producing labels very similar to classifier T

Fig. 4 Steps of an exploratory attack

3. The adversary uses the input data and the labels to train a deep learning classifier and optimize its hyperparameters to minimize the difference between the trained classifier and the target classifier.

The above attack trains a deep learning classifier that is functionally equivalent to (i.e. produces high-fidelity labels similar to) the classifier under attack. The functionally equivalent machine built by the adversary implicitly infers training data, type of the classifier and hyperparameters of the classifier. For testing, another set of input data is given to the classifier under attack and the classifier that is built by the adversary. The success of the adversary is measured by statistical similarity of labels returned by these two classifiers.

This attack results in classifier theft and poses a risk to the intellectual property of private companies and other organizations which have invested a significant amount of time and effort to gather training data for a classifier, to train the classifier and to tune its hyperparameters. If the classifier is stolen, then an organization's competitive advantage is reduced; in the long term, the possibility of classifier theft can discourage organizations from developing services based on machine learning techniques. Additionally, once a classifier is stolen, the adversary is free to analyse it (with an unlimited number of queries), in order to identify its potential weaknesses, as well as its underlying functionality. This can allow the adversary to subsequently launch an evasion attack against the original classifier, or to infer private information about the organization that developed it (such as its priorities and resources) or the underlying training data.

We consider an example of classification service on image data. A flower dataset [27] with images of daisies (class 1) and roses (class 2) is used to build the target classifier T. Note that this dataset has images of other flower types. Without loss of generality, we consider only daisy and rose images for the binary classification problem. This dataset has images in different picture formats and sizes (namely, different number of pixels). To build the same set of features for each image, we first scale all images to the same size and then change them to the common RGB format (i.e. each pixel has three values to be used as features). With these preprocessing, we can represent all images by the same number of features. Then, classifier T is built by applying deep learning with FNN on 638 training images and 636 test images.

In the black-box exploratory attack, the information of original classifier T is not available to the adversary. In addition, the adversary does not know the label of each sample. The adversary can only query some samples and collect the labels returned by T, and then use this information as training data to infer T by building a functionally equivalent classifier $\hat{T}$. Suppose that an adversary performs N queries for samples $s_1, s_2, \ldots, s_N$ and obtains $T(s_1), T(s_2), \ldots, T(s_N)$. For training and testing, N samples are split into two sets of data.

Deep learning is applied to build a classifier $\hat{T}$. The region of deep learning scores $\hat{\underline{r}}$ by classifier $\hat{T}$ is divided into subregions $\hat{\mathscr{R}}_i$ for $i \in \mathscr{C}$, such that if $\hat{\underline{r}}(s) \in \hat{\mathscr{R}}$ for a sample s, $\hat{T}(s) = i$. In a binary classification problem, deep learning provides a score within [0, 1] for each sample (i.e. $\hat{\underline{r}}$ is one-dimensional) and uses a threshold

$\hat{\theta}$ to classify samples. We aim to minimize the difference between $\hat{T}$ and T, which can be quantified as follows:

1. Average difference on class i:

$$d_i(T, \hat{T}) = \frac{\sum_{j \in \mathscr{C}}^{j \neq i} d_{ij}(T, \hat{T})}{\hat{n}_i}, \tag{9}$$

   where $d_{ij}(T, \hat{T})$ is the number of samples with $T(s) = i$ and $\hat{T}(s) = j$, and $\hat{n}_i$ is the number of samples in test data with $T(s) = i$.
2. Average overall difference:

$$d(T, \hat{T}) = \frac{\sum_{i \in \mathscr{C}} \sum_{j \in \mathscr{C}}^{j \neq i} d_{ij}(T, \hat{T})}{\sum_{i \in \mathscr{C}} \hat{n}_i}. \tag{10}$$

5 Evasion Attack

After the exploratory attack, the adversary can perform further attacks such as the *evasion attack* [21, 23], which aims to fool a target classifier into classifying selective input (test) data to the wrong output (label).

The preliminary step is to apply deep learning to infer the target classifier T and build a classifier $\hat{T}$ from the exploratory attack, which was discussed in the previous section. Using $\hat{T}$, we design an *evasion attack* by selecting a set of M samples to cause a large number of misclassifications from class i to class j by T (i.e. $C(s) = i$ and $T(s) = j$ for a sample s). Note that the objective can be changed to selecting a set of M samples to cause a large number of misclassifications (from any class to another one) by T (i.e. $C(s) \neq T(s)$ for many s). The goal of the adversary is to increase the misclassification error $e_{ij}(T)$ for given pairs of classes i and j or increase the overall misclassification error $\sum_i \sum_{j \neq i} e_{ij}(T)$ for all classes.

This evasion attack for given i and j follows three steps (shown in Fig. 5):

1. Apply classifier $\hat{T}$ on a set of test samples to obtain a deep learning score $\hat{\underline{r}}(s)$ on each sample s.
2. Determine an "attack region" of scores, $\hat{\mathscr{A}}_{ij}$, to cause misclassification from class i to class j.
3. Feed samples s with scores $\underline{r}(s) \in \hat{\mathscr{A}}_{ij}$ to the classifier T under attack.

The region $\hat{\mathscr{A}}_{ij}$ is defined by two conditions:

1. A sample s with $\underline{r}(s) \in \hat{\mathscr{A}}_{ij}$ is to be classified in class j by $\hat{T}$ (i.e. $T(s) = j$).
2. There is "significant" chance that s actually belongs to class i (i.e. $C(s) = i$).

Condition 1 means $\hat{\mathscr{A}}_{ij} \subseteq \hat{\mathscr{R}}_j$. Condition 2 can be interpreted as deep learning scores $\hat{\underline{r}}(s)$ being "close" to $\hat{\mathscr{R}}_i$. We need a distance measure $d(\underline{r}, \hat{\mathscr{R}})$ for a point $\underline{r}$

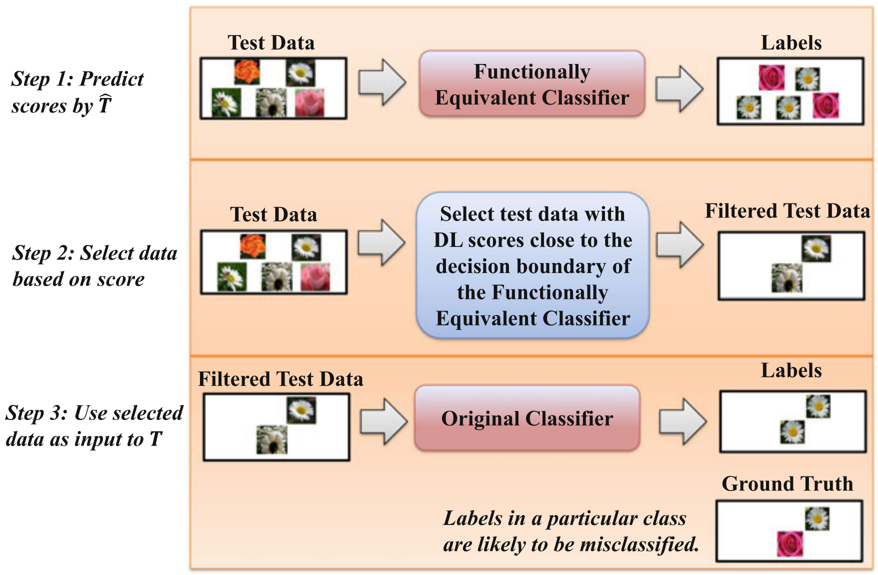
Step 1: Predict scores by $\widehat{T}$
Test Data
Functionally Equivalent Classifier
Labels
Step 2: Select data based on score
Test Data
Select test data with DL scores close to the decision boundary of the Functionally Equivalent Classifier
Filtered Test Data
Step 3: Use selected data as input to T
Filtered Test Data
Original Classifier
Labels
Ground Truth
Labels in a particular class are likely to be misclassified.

Fig. 5 Steps of an evasion attack

and a region $\hat{\mathscr{R}}$ in a high-dimensional space to define "close". For this purpose, we first define $d(\underline{r}, \underline{x})$ for two points as

$$\|\underline{r} - \underline{x}\|_2 = \sqrt{\sum_m (r_m - x_m)^2}, \tag{11}$$

where r_m and x_m are the mth element of $\underline{r}$ and $\underline{x}$, and define

$$d(\underline{r}, \hat{\mathscr{R}}) = \min_{\underline{x} \in \hat{\mathscr{R}}} d(\underline{r}, \underline{x}). \tag{12}$$

Then, "close" is defined by $d(\hat{\underline{r}}(s), \hat{\mathscr{R}}_i) \leq \phi$, where parameter $\phi > 0$. As a result, the attack region is characterized by

$$\hat{\mathscr{A}}_{ij} = \{\hat{\underline{r}}(s) : \hat{\underline{r}}(s) \in \hat{\mathscr{R}}_j, d(\hat{\underline{r}}(s), \hat{\mathscr{R}}_i) \leq \phi\}. \tag{13}$$

In the case that i and j are not given, the attack region is specified as the union of individual attack regions, namely $\bigcup_i \bigcup_{j \neq i} \hat{\mathscr{A}}_{ij}$.

In a binary classification problem, deep learning provides a score within [0, 1] for each sample by $\hat{T}$ and uses a threshold $\hat{\theta}$ to classify samples, i.e. $\hat{\mathscr{R}}_1 = [\hat{\theta}, 1]$ or $\hat{\mathscr{R}}_2 = [0, \hat{\theta}]$. Then, the attack region $\hat{\mathscr{A}}_{ij}$ for $i = 1$ and $j = 2$ in (13) is simplified to $\hat{\mathscr{A}}_{12} = [\hat{\theta} - \phi, \hat{\theta}]$.

Table 2 Error rate $e_{12}(T)$ for classifying image samples with scores from different attack regions

Attack region of scores	Error rate $e_{12}(T)$ (%)
[0, 0.3]	13.57
[0.3, 0.6]	29.17
[0.6, 0.8]	20.00
[0.8, 1]	4.32

For numerical results on image classification in the previous section, we evaluate the performance of the adversary (without knowing ground truth) pursues to fool the target classifier into misclassifying samples from class 1 as class 2. We assume $\phi = \frac{\hat{\theta}}{2}$ such that the attack region becomes $\hat{\mathscr{A}}_{12} = [\frac{\hat{\theta}}{2}, \hat{\theta}]$, which translates to the particular region $[0.3, 0.6]$. For comparison purposes, we also check the error probabilities if the adversary feeds the target classifier with images that have scores in $[0, 0.3]$, $[0.6, 0.8]$ and $[0.8, 1]$.

Results are shown in Table 2 for different attack regions. The error $e_{12}(T)$ on selected samples is the largest when we select images from the attack region $\hat{\mathscr{A}}_{12} = [0.3, 0.6]$. We conclude that evasion attack is more successful if it is launched in conjunction with an exploratory attack.

6 Extension of Evasion Attacks with Synthetic Data Inputs

A way of extending the evasion attack is generating *synthetic* data samples to be sent to the target classifier. In Sect. 5, we described how to select real data samples that are likely to be misclassified. To increase T's error rate, an adversary may examine a large number of samples by $\hat{T}$ and only select a small portion of them as input to T. As an extension, synthetic data can be also used in the evasion attack. This is an effective approach as many synthetic data samples can be generated and used as an input sample to T.

6.1 Evasion Attack with Synthetic Data Generated by the Generative Adversarial Network

A generative adversarial network (GAN) [3] can be used to generate synthetic data samples. There are two competing deep neural networks building the GAN structure:

1. The *generator network* G maps a source of noise to the input space and generates synthetic data samples, and
2. The *discriminator network* D distinguishes between received samples as synthetic or real.

The GAN trains the generator until it learns to fool the discriminator. Formally, the interactions between the generator and discriminator networks can be formulated as a *minimax game* problem:

$$\min_G \max_D \mathbb{E}_r \sim \mathbb{P}_{data}[\log(D(r))] + \mathbb{E}_z \sim \mathbb{P}_z[\log(1 - D(G(z)))],$$

where z is a noise input to generator G with a model distribution of $\mathbb{P}_z$ and $G(z)$ is the generator output. Input data r has distribution $\mathbb{P}_{data}$, and the discriminator D distinguishes between the real and generated samples. Both discriminator and generator networks are trained with backpropagation of error. We apply a deep convolutional GAN (DCGAN) to the flower dataset [16].

The discriminator takes image of size $140 \times 140 \times 3$ and feeds it to a three-layer cascade of convolutional layer, batch normalization and leaky ReLU activation function (leaky ReLU performs $f(x) = max(\alpha x, x)$ operation where α is the leakage, $\alpha = 0.2$). Each convolutional layer l has N_l distinct filters. We employed $N_1 = 64$, $N_2 = 128$ and $N_3 = 256$ filters. Each convolution filter has a size of 5×5 that performs convolution operation. The outputs of the convolutional layers are flattened and input to a fully connected layer with a single output which is activated by the sigmoid activation function. The generator of DCGAN takes in a noise of 200 dimensions and generates data that matches the statistics of the real data (in this case, daisy flowers). The generator is made up of four fractionally strided convolutions (performing the transpose of 2-D convolution filter) of 5×5 filter size and converts the noise representation into $140 \times 140 \times 3$ image. We train the DCGAN on the daisy flowers and generate synthetic images.

Examples of synthetic images (daisies) generated by GAN are shown in Fig. 6 for different numbers of iterations used to solve (14). These synthetic daisy images (class 1) are likely to be classified as rose (class 2). For instance, out of eight synthetic daisy images, two of them were misclassified as roses, while the original real image has a high classification score as daisy.

6.2 Evasion Attack with Synthetic Data Generated by Adversarial Perturbation

The evasion attack can be further strengthened by perturbing the selected real images and generating synthetic images that can be used to fool the classier under the evasion attack. While deep learning-based classifiers have been successfully applied to tasks such as image classification, recently they have been shown to be very unstable to adversarial perturbations of the data [3, 28–30], where an adversarial perturbation of a sample s is defined as the minimal perturbation $\Delta(s)$ that is sufficient to change the estimated label from $T(s)$ to $T(s + \Delta(s))$, i.e.

$$\Delta(s) = \min_\delta \|\delta\|_2 \tag{14}$$

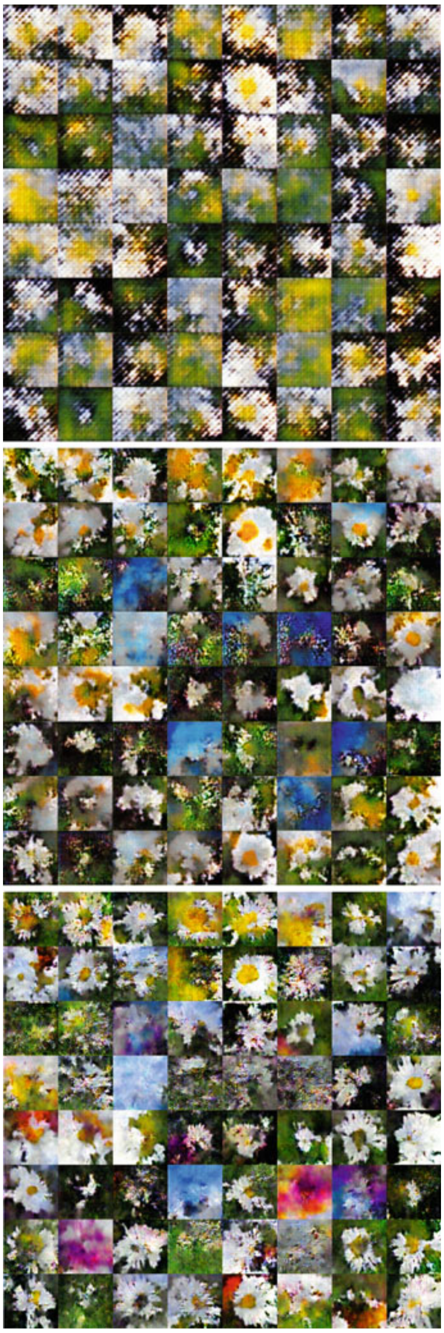
Fig. 6 Synthetic images (daisies) generated by GAN; top figure: synthetic images from iteration number 44; middle figure: synthetic images from the iteration number 88; bottom figure: synthetic images from the iteration number 177

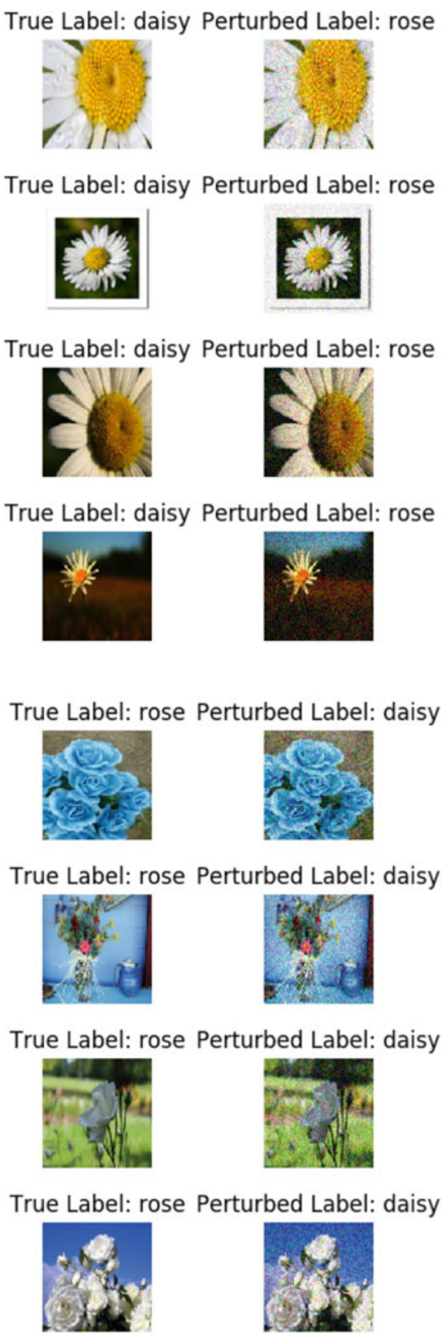

Fig. 7 Examples of mislabelled images by the classifier where original (left) and perturbed (right) images are shown side by side along with the labels of the classifier

subject to $T(s+\delta) \neq T(s)$. Similarly, the robustness of classifier T can be defined as

$$\rho_{adv} = \mathbb{E}_s \left[\Delta(s)/\|s\|_2 \right] , \tag{15}$$

where $\mathbb{E}_s$ is the expectation over distribution of data. With an adversarial perturbation of an image, the original image and the perturbed image can be labelled differently, which poses an important security problem. In [3], fast gradient sign method (FGSM) is proposed to efficiently compute adversarial perturbations for a given classifier. Even when the classifier is not fully known, we can use the estimated classifier. FGSM adds a perturbation vector

$$\delta = \varepsilon \, sign(\nabla_s J(\theta, s)) , \tag{16}$$

where ε is a small number, θ is the set of parameters of the neural network, and $J(\theta, s)$ is the cost used to train the neural network. The ε parameter perturbs an image such that it crosses over the decision boundary to be misclassified. If any adversarial attack detection mechanism is in place, this parameter needs to be finely tuned such that large perturbations may get detected and small perturbations may not be enough for misclassification. We apply the FGSM-based adversarial perturbation to the flower dataset [16]. The images are first resized to 140×140 dimensions and ε is taken as 0.2. A CNN is used. The first layer of convolutional filter consists of 64 filters with a kernel size of 8×8, which is followed by a layer of ReLU activation function to the output of the convolutional layer. The second and third convolutional layers also have 64 filters but with kernel sizes of 6×6 and 5×5, respectively. Both layers are also followed by a ReLU activation layer. The output of the third layer is flattened and input a fully connected layer to have N_{class} outputs, where N_{class} is the number of classes in the classifier. The classification accuracy of the classifier is 98%. When the perturbed images are presented to the classifier, the accuracy degrades to 40%. Figure 7 shows examples of the original images (left) and perturbed images (right) along with the labels of the classifier.

7 Defence Mechanisms

The analysis of vulnerability to adversarial machine learning presented in previous sections provides insights into the design of defence mechanisms. In general, defence mechanisms can be categorized into *proactive* and *reactive*.

1. In a *proactive* defence mechanism, the information system's goal is to increase the uncertainty that is observed by the adversary and prevent it from reliably inferring the classifier in an exploratory attack. Once the potential impact by an exploratory attack could be reduced, an evasion attack that is built upon inference results of the exploratory attack would not be effective, as well. One countermeasure against exploratory attack is adding some small but controlled perturbation (or

noise) to some selected output labels returned by the machine learning system [8]. Then, when the adversary polls the machine learning system, the training data collected by the adversary becomes noisy. Since deep learning is sensitive to training errors, the adversary cannot reliably train a deep neural network to infer the target machine learning system. How to insert errors in the returned labels is a delicate task. While inserting a very small number of errors would not be effective, a large number of errors would make the normal operation of machine learning system unreliable. One way of solving this problem is introducing errors to labels that have high confidence scores (i.e. labels of data samples that are more likely to be correctly classified) [8]. With this approach, it is difficult for the adversary to train a deep neural network, since the decision boundaries cannot be constructed reliably because of the errors in these dominant labels that are instrumental in shaping the decision boundaries.

2. In a *reactive* defence mechanism, the information system's goal is to identify which users are attacking the machine learning system and filtering out or restricting their queries. One potential indicator of an evasion attack is that a particular user is always sending input data samples that have the classification scores close to the decision boundary. These data samples are more likely to be misclassified than others. Therefore, such requests could be flagged. A similar approach was followed in [25] for a causative attack, where user inputs are monitored regarding their consistency with respect to the existing classifier that is being trained with additional input data from users.

8 Extension of Adversarial Deep Learning to Other Domains

Adversarial deep learning has applications not only limited to cybersecurity but also available in other domains that take inputs for decision making and provide outputs either directly to the user or in a form that can be observed by third parties. Vulnerability detection and analysis originally considered for cybersecurity systems can be reused or tailored for these new domains.

One extension of adversarial deep learning is to the area of *wireless network security*. Machine learning finds numerous applications in wireless communications such as Wi-fi, LTE, sensor networks and mobile ad hoc networks (MANETs). One canonical example is the cognitive radio [31] that can be programmed and configured dynamically by automated means to use the best wireless channels in its vicinity to avoid or intelligently manage user interference and congestion. The design concepts of cognitive radios have become a reality with the emergence of low-cost software-defined radios (SDRs) that can perform various sensing, communications and networking tasks [32]. The applications of machine learning to cognitive radio span the entire network protocol stack including automated detection, classification and prediction tasks such as spectrum sensing, signal classification, modulation recognition and adaptation, dynamic spectrum access (DSA), power control, routing and flow control. Machine learning can be used to perform these tasks, thereby supporting cognitive radios to perceive and learn the spectrum environment and adapt to spectrum dynamics. One example is sensing the spectrum, i.e. classifying whether the spectrum is used by other transmitter(s), or not. For instance, the GAN has been applied in [33] to support spectrum sensing with additional training data.

Due to the broadcast nature of wireless medium, wireless networks are susceptible to frequent attacks and exploits such as jamming attacks. Cognitive radios aim to detect and mitigate these attacks and exploits by automated means such as those enabled by machine learning. In this process, cognitive radio observes the spectrum (including the effects of adversaries such as jammers) and takes actions for communications (such as deciding to transmit or wait). These actions are potentially observed by adversaries that can react to them.

In this context, the adversary, e.g. a jammer, has the opportunity to follow the design constructs of adversarial machine learning to launch wireless network attacks. Building upon the insights gained from information system security, the following procedure has been applied in [34] to launch wireless attacks based on adversarial machine learning. The adversary launches an exploratory attack by training a classifier that predicts when the cognitive radio will transmit next (or better, when there will be the next successful transmission). To train the classifier, the adversary builds the training data by sensing the spectrum and tracking the transmissions of the

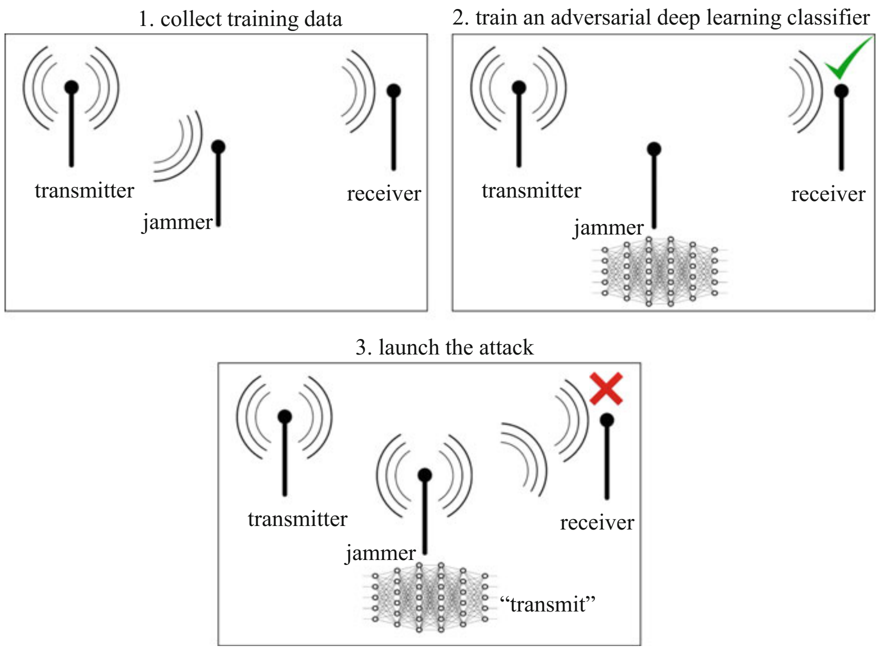

Fig. 8 Application of adversarial deep learning to wireless security domain

cognitive radio over time with respect to some spectrum features, such as the perceived availability of the spectrum. Once the classifier is trained, the jammer can predict the future behaviour of the cognitive radio (i.e. when it will transmit or when it will transmit successfully) and aims to jam the signal of the cognitive radio based on the prediction results obtained in the exploratory attack. This procedure is illustrated in Fig. 8. Applications of adversarial machine learning are broad. Similar use cases can be found in other domains such as Internet of Things, cyberphysical systems and autonomous driving.

9 End-of-Chapter Teaching Questions

1. Suppose a classifier T on two classes 1 and 2 has the following error probabilities: $e_1(T) = 10\%$ and $e_1(T) = 20\%$. The number of samples in each class is $n_1 = 200$ and $n_2 = 300$. Calculate the overall error probability.
2. What are the two non-deep learning classifiers discussed in this chapter? What are the assumptions made by these classifiers?
3. What are different types of deep neural networks? What is the difference among them?
4. What are the three types of adversarial deep learning attacks? Among them, which attack changes the original classifier?
5. Do we need to know the actual class information (i.e. $C(s)$) in an exploratory attack? If not, which information is used as label to train the inferred classifier $\hat{T}$?
6. Assuming that an adversary has no knowledge of the actual class information (i.e. $C(s)$), can an adversary measure the increased error rate due to the evasion attack?
7. What are ways of generating synthetic data for the evasion attack?
8. What is the benefit to apply the GAN in the evasion attack?
9. What are the two types of defence mechanisms discussed in this chapter? What is the difference between them?
10. If we apply deep learning to spectrum sensing for cognitive radios, what are the two possible classes?

10 End-of-Chapter Research Questions

1. What are the examples of online machine learning systems that are potentially susceptible to adversarial machine learning attacks?
2. What types of machine leaning tasks (other than classification) are potentially vulnerable to adversarial machine learning?
3. What are the other domains in which adversarial machine learning may find applications?
4. What are the types of deep learning algorithms that can be used for different data domains?

5. How can the adversary launch other (new) types of attacks built upon an exploratory attack?
6. What is the computational complexity of adversarial deep learning and how can it be improved, if needed?
7. How can one parallelize adversarial deep learning-based attacks and the corresponding defence mechanisms?
8. What types of software, firmware and hardware platforms are needed to launch effective adversarial deep learning-based attacks and deliver countermeasures?
9. What are the other defence mechanisms that can be used against adversarial deep learning?
10. What are the potential ways (if any) to explain the decisions of adversarial deep learning?

References

1. Stratosphere IPS (2018). https://www.stratosphereips.org. Accessed 15 Mar 2018
2. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al (2016) Mastering the game of Go with deep neural networks and tree search. Nature **529**(7587), 484–489
3. Goodfellow IJ, Shlens J, Szegedy C (2014) Explaining and harnessing adversarial examples. arXiv:1412.6572
4. Huang L, Joseph AD, Nelson B, Rubinstein BI, Tygar J (2011) Adversarial machine learning. In: Proceedings of the 4th ACM workshop on security and artificial intelligence. ACM, pp 43–58
5. Miller B, Kantchelian A, Afroz S, Bachwani R, Dauber E, Huang L, Tschantz MC, Joseph AD, Tygar JD (2014) Adversarial active learning. In: Proceedings of the 2014 workshop on artificial intelligent and security workshop. ACM, pp 3–14
6. Laskov P, Lippmann R (2010) Machine learning in adversarial environments. Springer, Berlin
7. Shi Y, Sagduyu Y, Grushin A (2017) How to steal a machine learning classifier with deep learning. In: 2017 IEEE international symposium on technologies for homeland security (HST). IEEE
8. Shi Y, Sagduyu YE (2017) Evasion and causative attacks with adversarial deep learning. In: MILCOM 2017-2017 IEEE military communications conference (MILCOM). IEEE, pp 243–248
9. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. In: Advances in neural information processing systems, pp 2672–2680
10. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2(5):359–366
11. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
12. LeCun Y et al (1989) Generalization and network design strategies. Connectionism in perspective, pp 143–155
13. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
14. Rumelhart DE, Hinton GE, Williams RJ (1985) Learning internal representations by error propagation. Technical report, University of California, San Diego, La Jolla, Institute for Cognitive Science
15. Microsoft Cognitive Toolkit (CNTK) (2018). https://docs.microsoft.com/en-us/cognitive-toolkit. Accessed 15 Mar 2018

16. Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M et al (2016) Tensorflow: large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467
17. Barreno M, Nelson B, Sears R, Joseph AD, Tygar JD (2006) Can machine learning be secure? In: Proceedings of the 2006 ACM symposium on information, computer and communications security. ACM, pp 16–25
18. Ateniese G, Mancini LV, Spognardi A, Villani A, Vitali D, Felici G (2015) Hacking smart machines with smarter ones: how to extract meaningful data from machine learning classifiers. Int J Secur Netw 10(3):137–150
19. Fredrikson M, Jha S, Ristenpart T (2015) Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM, pp 1322–1333
20. Tramèr F, Zhang F, Juels A, Reiter MK, Ristenpart T (2016) Stealing machine learning models via prediction APIs. In: USENIX security symposium, pp 601–618
21. Biggio B, Corona I, Maiorca D, Nelson B, Šrndić N, Laskov P, Giacinto G, Roli F (2013) Evasion attacks against machine learning at test time. In: Joint European conference on machine learning and knowledge discovery in databases. Springer, Berlin, pp 387–402
22. Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A (2016) Practical black-box attacks against deep learning systems using adversarial examples
23. Kurakin A, Goodfellow I, Bengio S (2016) Adversarial examples in the physical world. arXiv:1607.02533
24. Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A (2016) The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). IEEE, pp 372–387
25. Pi L, Lu Z, Sagduyu Y, Chen S (2016) Defending active learning against adversarial inputs in automated document classification. In: 2016 IEEE global conference on signal and information processing (GlobalSIP). IEEE, pp 257–261
26. Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A (2017) Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on asia conference on computer and communications security. ACM, pp 506–519
27. Flower Image Dataset (2018). https://www.tensorflow.org/tutorials/image_retraining. Accessed 15 Mar 2018
28. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2013) Intriguing properties of neural networks. arXiv:1312.6199
29. Nguyen A, Yosinski J, Clune J (2015) Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 427–436
30. Moosavi Dezfooli SM, Fawzi A, Frossard P (2016) Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of 2016 IEEE conference on computer vision and pattern recognition (CVPR)
31. Haykin S (2005) Cognitive radio: brain-empowered wireless communications. IEEE J Sel Areas Commun 23(2):201–220
32. Soltani S, Sagduyu Y, Shi Y, Li J, Feldman J, Matyjas J (2015) Distributed cognitive radio network architecture, SDR implementation and emulation testbed. In: MILCOM 2015-2015 IEEE military communications conference. IEEE, pp 438–443
33. Davaslioglu K, Sagduyu YE (2018) Generative adversarial learning for spectrum sensing. In: Accepted to IEEE international conference on communications (ICC). IEEE
34. Shi Y, Sagduyu YE, Erpek T, Davaslioglu K, Lu Z, Li JH (2018) Adversarial deep learning for cognitive radio security: jamming attack and defense strategies. In: IEEE international communications conference workshop on promises and challenges of machine learning in communication networks. IEEE