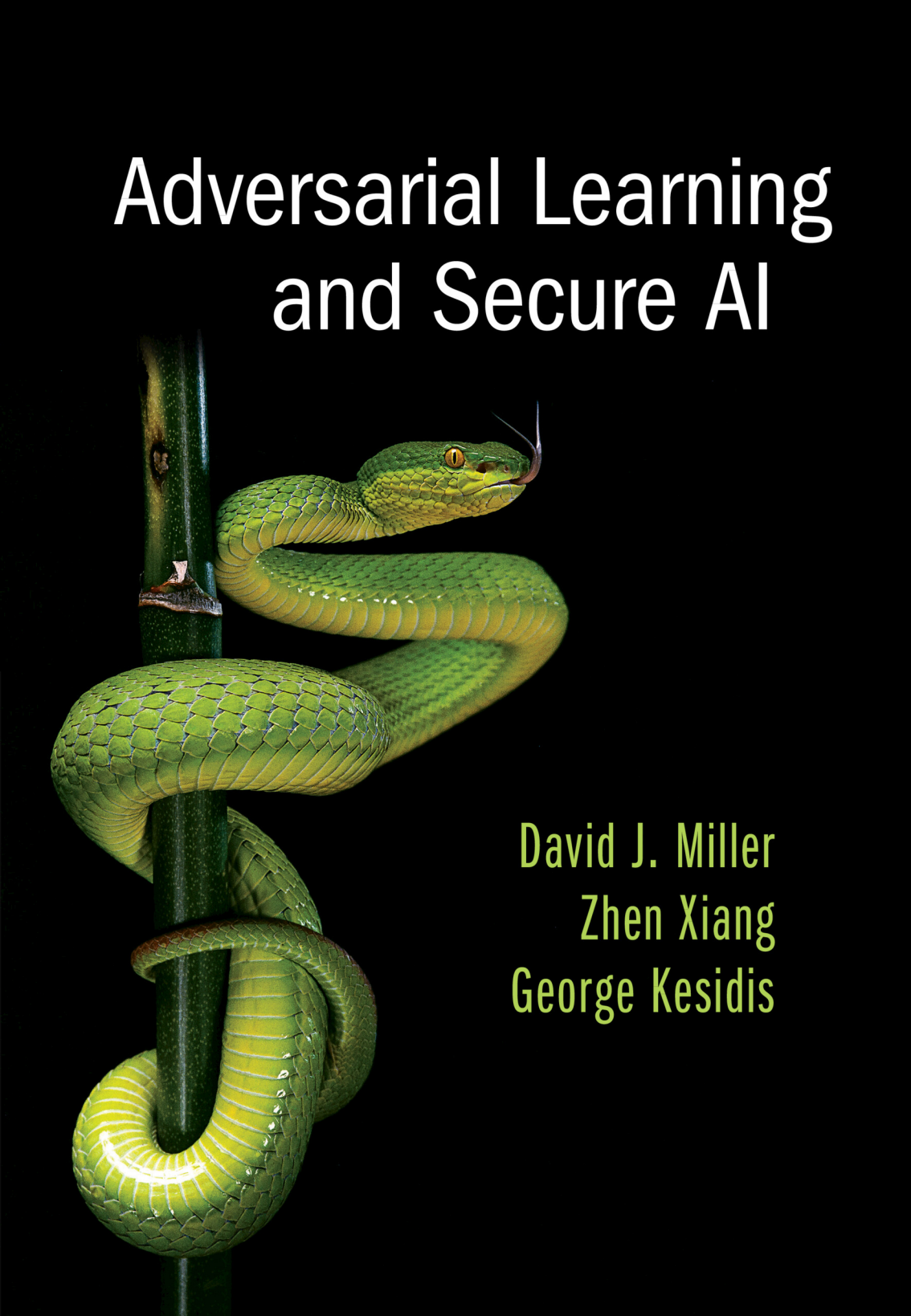
Adversarial Learning and Secure AI



David J. Miller
Zhen Xiang
George Kesidis

Adversarial Learning and Secure AI

Providing a logical framework for student learning, this is the first textbook on adversarial learning. It introduces students to attacks and vulnerabilities of deep learning, and to methods for defending against attacks and making AI generally more robust.

It is the ideal resource for upper undergraduate and first-year graduate courses on AI security and adversarial learning. Students and instructors will benefit from these features

- application examples, case studies, and real-world student projects in each chapter, connecting theory with practice
- a project-driven approach that strengthens critical thinking when evaluating attacks and defenses
- a variety of application areas covered by the examples and projects, for example, image classification, text classification, point cloud classification, and a regression example from finance.

David J. Miller is professor of Electrical Engineering at the Pennsylvania State University.

Zhen Xiang is a post-doctoral research associate of Computer Science at the University of Illinois, Urbana-Champaign.

George Kesidis is a professor of Computer Science and Engineering, and of Electrical Engineering, at the Pennsylvania State University.

"This textbook is one of the first major efforts to systematically examine adversarial machine learning. It clearly outlines the most common types of attacks on machine learning/AI, and defenses, with rigorous yet practical discussions. I would highly recommend it to any instructor or machine learning student who seeks to understand how to make machine learning more robust and secure."
Carlee Joe-Wong, Carnegie Mellon University

"This is a clear and timely introduction to the vital topic of adversarial learning. As leading international experts, the authors provide an accessible explanation of the foundational principles and then deliver a nuanced and extensive survey of recent attack and defense strategies. Multiple suggested projects allow the book to serve as the core of a graduate course."
Mark Coates, McGill University

"Remarkably comprehensive, this book explores the realm of adversarial learning, revealing the vulnerabilities and defenses associated with deep learning. With a mix of theoretical insights and practical projects, the book challenges the misconceptions about the robustness of Deep Neural Networks, offering strategies to fortify them.
It is well suited for students and professionals with basic calculus, linear algebra, and probability knowledge, and provides foundational background on deep learning and statistical modeling. A must-read for practitioners in the machine learning field, this book is a good guide to understanding adversarial learning, the evolving landscape of defenses, and attacks."
Ferdinando Fioretto, Syracuse University

"In a field that is moving at break-neck speed, this book provides a strong foundation for anyone interested in joining the fray."
Amir Rahmati, Stony Brook

Adversarial Learning and Secure AI

DAVID J. MILLER
Pennsylvania State University

ZHEN XIANG
University of Illinois, Urbana-Champaign

GEORGE KESIDIS
Pennsylvania State University

CAMBRIDGE
UNIVERSITY PRESS

Shaftesbury Road, Cambridge CB2 8EA, United Kingdom

One Liberty Plaza, 20th Floor, New York, NY 10006, USA

477 Williamstown Road, Port Melbourne, VIC 3207, Australia

314–321, 3rd Floor, Plot 3, Splendor Forum, Jasola District Centre, New Delhi – 110025, India

103 Penang Road, #05–06/07, Visioncrest Commercial, Singapore 238467

Cambridge University Press is part of Cambridge University Press & Assessment, a department of the University of Cambridge.

We share the University's mission to contribute to society through the pursuit of education, learning and research at the highest international levels of excellence.

www.cambridge.org

Information on this title: www.cambridge.org/highereducation/isbn/9781009315678

DOI: 10.1017/9781009315647

First published 2024

Printed in the United Kingdom by CPI Group Ltd, Croydon, CR0 4YY, 2024

A catalogue record for this publication is available from the British Library

A Cataloging-in-Publication data record for this book is available from the Library of Congress

ISBN 978-1-009-31567-8 Hardback

Additional resources for this publication at www.cambridge.org/millersecureAI.

DJM dedicates this book to his children Joshua and Madeline

ZX dedicates this book to his son Ian

GK dedicates this book to Fozzie, Gonzo and Therese

The authors also share a dedication of this book to their collaborators Xi Li and Hang Wang

Contents

Preface

Why We Wrote This Book

In the past ten years, deep learning has been applied to many market and government sectors (e.g., health, finance, military, intelligence, manufacturing, sales), including in their critical infrastructure and supply chains (MLOps/AIOps). Application domains include those where operational conditions may change over time (model drift), where safety and security are of great concern, and where significant financial stakes are involved. As such, the deep learning process and the trained Deep Neural Networks (DNNs or "AIs") themselves have become targets of attack. More generally, basic questions about the robustness and explainability of DNN solutions have also been raised even in the absence of attacks, for example, [132]. A research sub-field assessing and addressing the risks associated with using AIs (and other machine learning models) is known as *adversarial learning*. This area essentially represents a merger between the fields of computer security and machine learning.

An important aspect of software security is to consider how the software will behave for *all possible* valid inputs. The reason for this is that an adversary may exploit a vulnerability that pertains to a range of inputs for which the software's behavior was not carefully considered by its developers. This is a daunting security task for a DNN, whose behavior depends on an enormous set of parameters (even billions) which are heuristically learned, and whose input space may be very high-dimensional. What this means is that DNNs have a substantial attack "surface," which makes them vulnerable to a variety of attacks/exploits. While some basic adversarial learning research dates back more than 20 years, this field really took off with the observation in 2014 that *adversarial inputs* may be easily constructed for DNNs – these are small changes to an input pattern, *imperceptible* to a human being, and yet which greatly alter the DNN's output (e.g., changing its class decision). Aside from being a security threat, adversarial inputs demonstrate that it is a fallacy – held by many researchers, educators, industrialists, and journalists – that DNNs are generally robust, reliable decision-makers, and are close to fulfilling the promise of artificial intelligence. In the year 2020 alone, more than 1100 papers on adversarial learning were submitted to arXiv.org. While there are a number of review papers, to date there are no books on this subject which are suitable for a course offering.

The Emphasis of the Book is Unsupervised Defenses

Generally, defenses and attacks continuously evolve. New vulnerabilities are discovered by attackers (either in the system being protected or in its defenses) and exploited. Defenses may evolve to address newly identified vulnerabilities (including those recently revealed by new exploits). It may be unrealistic to suppose that the defender has detailed knowledge of an attack that may be mounted. This is why we focus on unsupervised defenses that aim to protect against a whole family of attacks (rather than relying on somehow obtained knowledge of a specific *known* attack [269, 270])[1]. On the other hand, in the quest for the glory (and concomitant research funding) associated with finding a new vulnerability and devising an exploit for it, some researchers ignore existing or obvious defenses which would be effective against their attacks, or get carried away and unrealistically assume an omniscient or omnipotent attacker (e.g., one who completely controls the training dataset and training process, or controls how new samples are labeled in an active learning context). Given an omnipotent adversary, a defense may be able to do little more than increase the attacker's work factor. That said, though "security through obscurity" is commonly practiced and may be effective in some cases, assuming some attacker knowledge of a defense is not unreasonable. This is especially true considering spectacular leaks by insiders and breaches in privacy protections in the recent past.[2]

Purpose, Target Audience and Prerequisites

The targeted audience for this book is senior undergraduates and graduate students in all branches of science and engineering. The purpose of this book is to introduce students to existing attacks and vulnerabilities of deep learning (and machine learning in general) and to methods for defending against these attacks, as well as for making AI generally more robust (even in the absence of attack). Along the way, students will also enhance their appreciation for what deep neural networks are in fact learning (and what they are not learning). For example, students will learn that training dataset augmentation (i) may improve generalization performance, (ii) instead, may cause degradation in DNN accuracy (e.g., by overfitting through adversarially robust learning), (iii) or may result in the planting of a backdoor in the DNN. As another example, students will better understand the circumstances in which DNNs learn patterns that are spatially invariant (occurring anywhere in an image), or only patterns that are spatially fixed. The book covers many attack-defense scenarios and involves many case studies and real-world problems addressed by the state-of-the-art in recent research publications.

[1] Note that an antivirus system or firewall typically functions in response to known attacks, that is, they are supervised defenses. Hence periodic updates (with patching of exposed vulnerabilities) are needed, typically *after* a new exploit has been detected and carefully studied to identify its signature.

[2] Which can go both ways, that is, new attacks can also be leaked before they are launched; but, to reiterate, we focus on unsupervised defenses herein.

Prerequisites for this book include a basic introduction to calculus, linear algebra, and probability. Though the second and third chapters provide the necessary background material on deep learning, detection, and statistical modeling, a student would benefit from a more broadly scoped course on pattern recognition and machine learning based on, for example, [63, 190], and from an introduction to numerical analysis, for example, [8].

Projects

There are course projects at the ends of the chapters that give hands-on experience to students in devising and evaluating both attacks and defenses against machine learning systems. These projects are intended as the primary homework exercises for a course on robust and adversarial learning. They also serve the dual purpose of helping students to obtain familiarity and facility in machine learning design within the Python programming environment (in particular, the use of PyTorch for deep learning). Moreover, these projects provide a window for students into how much research work is being conducted in AI/machine learning – with promising new ideas postulated and then experimentally assessed, both to validate (or reject) them and to obtain greater insight into the problem at hand. Given some Python experience, students can learn PyTorch [209] while studying the first few chapters of this book. Also, a tutorial on the Pillow fork of the Python Image Processing Library (PIL) will be useful, for example, [88]. PyTorch code for projects given at the end of Chapters 4, 5, 6 and 13 is available at: www.cambridge.org/millersecureAI.

Quite a bit of code is provided for the first few preliminary PyTorch projects (the provided code should be carefully studied by the student), while little to no code is provided for subsequent projects. The idea is that the students can "fill in the blanks" for the first PyTorch projects that are assigned but have to produce all of the code for subsequent ones.

Chapter Roadmap

The first three chapters respectively provide background on attack types and attack nomenclature, on deep learning, and on detection and estimation. If students have taken a prior course on pattern recognition or machine learning, they may be able to skip Chapters 2 and 3. Note that subsequent chapters frequently refer back to material in Chapters 2 and 3.

Chapter 4 addresses defenses against adversarial inputs at test-time, also known as test-time evasion (TTE) attacks.

Chapter 13 addresses defense against general data poisoning attacks against classifiers.

Chapter 14 addresses defense against reverse-engineering (probing) attacks.

A road map for the remaining chapters on backdoor defense is as follows.

- Chapter 5 addresses backdoor defense implemented by the training authority, who has access to the (possibly poisoned) training set and who controls the training process.
- The next four chapters address *post-training* backdoor defense, where the defender does *not* in fact have access to the training set, but only to the trained classifier and to a (relatively) very small set of clean (unpoisoned) labeled samples.
- Chapter 6 addresses defense against imperceptible backdoor attacks. One approach reverse-engineers putative backdoor patterns that are additively incorporated either to the input (raw features) or to an internal layer of the neural network (embedded features). The reverse-engineered backdoor pattern has utility beyond post-training detection (it can also be used for test-time detection of backdoor triggers and for mitigating the effect of backdoors). Moreover, it is an important element of explainable AI (XAI), indicating patterns in the presence of which a DNN's decision-making is fragile.
- Chapter 7 addresses post-training defense against backdoors that are embedded by replacing a "patch" of input features by the backdoor pattern. These backdoor attacks can be implemented either digitally or *physically* (e.g., by placing an object – the backdoor pattern – in a given scene). One reverse-engineering defense exploits the fact that the attack should be "scene-plausible" in order to be evasive.
- The defenses in Chapters 6 and 7 are not very suitable when the number of classes in the problem is small (e.g., the two-class case), since in this case there are insufficient detection statistics available for estimating the parameters that specify a detection rule. The post-training defenses in Chapter 8 address this problem. The main defense developed there was found to be effective with a *constant* detection threshold ($\frac{1}{2}$), *irrespective* of the DNN architecture and classification domain.
- Chapter 9 considers defenses that aim to be *universal*, that is, without any explicit or implicit assumptions about the backdoor pattern or how it was embedded.
- Chapter 10 considers "in-flight" detection, that is, detection of backdoor triggers in input patterns at test time. Such detection may give the potential to catch culprits in the act of exploiting the backdoor mapping. One such described defense leverages a reverse-engineered backdoor pattern.
- Chapter 11 considers backdoor detection for non-image point cloud data classifiers.
- Chapter 12 considers backdoors for regression rather than classification and also discusses active learning.

The authors acknowledge the support of students and colleagues. In particular, we thank Yujia Wang (Chapters 4 and 14), Xi Li (Chapters 10, 12 and 13), and Hang Wang (Chapters 4 and 9), as well as Zhicong Qiu and Xinyi Hu. We also thank Vladimir Lucic for consultations regarding Chapter 12.

The authors acknowledge the sources of our research support through the Pennsylvania State University: an AFOSR DDDAS grant (2017–2021), an ONR NROTC education grant (2021–2022), an NRC Research Associate Fellowship with AFRL (2021–2022), and two Cisco Systems gifts (2019, 2022). Through Anomalee Inc., we also acknowledge the research support of an NSF SBIR Phase-One grant (2022–2023).

Notation

Typically,

- random objects are denoted by capital (upper-case) letters
- non-vector matrices are denoted by bold capital letters, for example,

$$\mathbf{V} = [v_{i,j}]_{i=1,\ldots,n,\ j=1,\ldots,m}$$

  denotes an $n \times m$ matrix with entry $v_{i,j}$ in the ith row and jth column, and both $m > 1$ and $n > 1$
- column vectors are denoted by underlined lower-case letters
- datasets are typically denoted by calligraphic capital letters
- some variables not defined below, such as $x, y, z, n, m, i, j, k, \alpha, \beta, \kappa, \theta$, are often repurposed in various chapters
- some symbols, such as f, g, are typically used for functions and are also often repurposed

More specifically, we define the following mathematical symbols and operators

- $\mathbb{R}$ is the set of real numbers
- $\mathbb{Z}$ is the set of integers
- $\mathbb{Z}^+$ is the set of positive integers (natural numbers)
- N is the dimension of the input sample space (space of input patterns) of a feed-forward neural network, that is, the space of N-dimensional, real-valued column vectors, $\mathbb{R}^N$
- $\underline{z}'$ is the transpose of column vector $\underline{z}$, that is, $\underline{z}'$ is a row vector
- $\langle \underline{z}, \underline{y} \rangle = \underline{z}'\underline{y} = \sum_{j=1}^{N} z_j y_j$ is the inner (dot) product of (column) vectors $\underline{z}, \underline{y} \in \mathbb{R}^N$
- $\|\underline{x}\|_q = (\sum_{i=1}^{N} x_i^q)^{1/q}$ is the l_q-norm (or q norm) of vector $\underline{x} \in \mathbb{R}^N$
- $\|\underline{x} - \underline{y}\|_q$ is the l_q distance between $\underline{x}$ and $\underline{y}$ of the same dimension
- $\|\underline{x}\| = \|\underline{x}\|_2 = \sqrt{\underline{x}'\underline{x}} = \sqrt{\langle \underline{x}, \underline{x} \rangle}$ is the Euclidean (l_2) norm of $\underline{x}$
- $\underline{x} \odot \underline{m}$ is element-wise multiplication of the vectors (or matrices) $\underline{x}, \underline{m}$ resulting in another vector (or matrix), that is the ith element of $\underline{x} \odot \underline{m}$, $(\underline{x} \odot \underline{m})_i = x_i m_i$
- $\mathcal{X}$ is the set of data samples that are used for training a neural network (deep learning), where $\mathcal{X} \subset \mathbb{R}^N$
- $T = |\mathcal{X}| < \infty$ is the number of samples in the dataset $\mathcal{X}$
- K is the (finite) number of classes in $\mathcal{X}$ for classification problems (but K has different meaning in the context of K-means clustering or KNN classification)

- $\mathcal{Y}$ is the set of classes in $\mathcal{X}$, that is, $K = |\mathcal{Y}|$, for example, $\mathcal{Y} = \{1, 2, \ldots, K\}$
- $c(\underline{x}) \in \mathcal{Y}$ is the true class label of $\underline{x} \in \mathbb{R}^N$
- $\hat{c}(\underline{x})$ is the inferred class of input sample $\underline{x}$ by a classifier
- $\mathsf{E}X = \mathsf{E}(X) = \mathsf{E}[X]$ is the expectation of random variable X
- $\mathsf{P}(A) = \mathsf{P}[A]$ is the probability of event A
- $\{x_1, x_2, \ldots, x_n\}$ is a set with n elements
- $\{x_a \,|\, a \in A\} = \{x_a : a \in A\}$ is the set of elements x_a *such that* (: or |) parameter or index a belongs to the set A (here $x(a)$ or $x^{(a)}$ may be used instead of x_a to indicate the dependence of x on a)
- $A \cup B$ and $A \cap B$ respectively are the union and intersection of the sets A and B
- $A \backslash B$ is the set of elements in the set A that are not in the set B
- $\emptyset$ is the empty set
- $[a, b) = \{r \in \mathbb{R} : a \leq r < b\}$, with $b > a$, is a contiguous interval of real numbers including a but not b
- $\mathbf{1}\{\xi\} = \mathbf{1}(\xi)$ is an indicator function, equal to one if the statement ξ is true and zero if ξ is false
- $\mathbf{I}$ is a square identity matrix, with 1s on the diagonal and 0s off diagonal
- $a := b$ or $a \triangleq b$ means a equals b by definition
- $\underline{0}$ is a vector all of whose entries are zero
- $\underline{1}$ is a vector all of whose entries are one
- $\underline{X} \sim F$ means random vector $\underline{X}$ has (multivariate) distribution F
- $\det(\mathbf{A}) = |\mathbf{A}|$ is the determinant of square matrix $\mathbf{A}$
- Δx is a change in the quantity x

List of Acronyms

- 3D: three-dimensional
- ACC: Accuracy (on a clean test/evaluation set)
- AD: Anomaly Detection (short name for I-PT-RED in Chapter 6)
- AI: Artificial Intelligence (often synonymous with a DNN)
- AL: Active Learning
- a.s.: almost surely (with probability one)
- ASR: Attack Success Rate
- AUC: Area Under the (ROC) Curve
- BA: Backdoor Attack (Trojan)
- BIC: Bayesian Information Criterion
- BP: Backdoor Pattern
- CDF or cdf: Cumulative Distribution Function
- CNN: Convolutional Neural Network
- CS: Cosine Similarity
- DNN: Deep Neural Network
- DP: Data Poisoning (attack)
- ET: Expected Transferability

- FPR: False Positive Rate (fraction or percentage)
- GAN: Generative Adversarial Network
- GMM: Gaussian Mixture Model
- HC: High Confidence
- i.i.d.: independent and identically distributed
- JSD: Jensen–Shannon Divergence
- KL: Kullback–Leibler divergence
- *K*NN: *K* Nearest Neighbors
- LC: Low Confidence
- LEM: Local Error Maximizer
- LeNet-*n*: Learnable Neural Network architecture with *n* layers [140]
- LR: Logistic Regression
- LSTM: Long Short-Term Memory (a recurrent NN)
- MAD: Median Absolute Deviation
- MAE: Mean Absolute Error
- MAP: Maximum a posteriori
- ML: Machine Learning
- MLE: Maximum Likelihood Estimation
- MM: Mixture Model (or Maximum Margin in Chapter 9)
- MSE: Mean-Squared Error
- NB: Naive Bayes
- NN: Neural Network
- OOD: Out-Of-Distribution
- OODD: Out-Of-Distribution Detection
- pAUC: partial (ROC) Area Under the Curve
- PCA: Principal Component Analysis
- pdf: probability density function
- pmf: probability mass function
- PMM: Parsimonious Mixture Modeling [86]
- PT: Post-Training
- RE: Reverse-Engineering
- RE-AP: Reverse-Engineering Additive Perturbation
- RE-PR: Reverse-Engineering Patch Replacement
- REA: Reverse-Engineering Attack
- RED: Reverse-Engineering Defense
- ResNet-*n*: Residual Neural Network architecture with *n* layers [97]
- RL: Reinforcement Learning
- ROC: Receiver Operating Characteristic
- SGD: Stochastic Gradient Descent
- SIA: Source-class Inference Accuracy
- SVD: Singular Value Decomposition
- SVM: Support Vector Machine
- TPR: True Positive Rate (fraction or percentage)
- TSC: Training Set Cleansing

- TTE: Test-Time Evasion (attack), that is, adversarial input
- WB: White Box
- XAI: eXplainable AI

The following list contains the "proper names" of some attacks and defenses used in this book, with bibliographic citations

- AC-GAN: Auxiliary-Classifier GAN based TTE detection [284, 285]
- ADA: Anomaly Detection of TTE Attacks [179]
- B3D: Black Box Backdoor trigger Detection [61]
- BIC-MM-TSC: BIC-MM based TSC against error generic DP [148]
- CI: Cluster Impurity defense [308]
- CIFAR-n: Canadian Institute for Advanced Research color image dataset with n object classes [129]
- CW: Carlini–Wagner TTE attack [33]
- FGSM: Fast Gradient Sign Method for TTE attacks [83]
- FP: Fine Pruning backdoor defense [156]
- i-FGSM or BIM: iterative-FGSM or Basic Iterative Method for TTE attacks [133]
- I-PT-RED: Imperceptible–backdoor PT-RED [303, 307]
- IF-RED: In-Flight backdoor trigger RED [149]
- JSMA: Jacobian based Saliency Map Approach for TTE attacks [203]
- KD: Kernel Density based defense [68]
- L-PT-RED: Lagrangian PT-RED [305]
- MD: Mahalanobis Distance based defense [142]
- MNIST: Modified National Institute of Standards and Technology dataset of handwritten digits [141]
- NC: Neural Cleanse backdoor detection [282]
- NC-M: NC based backdoor Mitigation [282]
- P-PT-RED: Perceptible backdoor PT-RED [304]
- PC-PT-RED: Point Cloud PT-RED against backdoors [306, 309]
- PGD: Projected Gradient Descent for TTE attacks [271]
- STRIP: STRong Intentional Perturbation backdoor trigger detection [73]
- T-PT-RED: Transferable PT-RED against backdoor DP [302]
- TSC-RED: Training dataset Cleansing RED against backdoor DP [301]
- UnivBD: "Universal" Backdoor Detection approach [286]
- UnivBM: "Universal" Backdoor Mitigation approach [286]
- ZOO: Zeroth Order Optimization based TTE attacks [41]

1 Overview of Adversarial Learning

In this chapter, we introduce attacks/threats against machine learning systems. Attacks that will be covered in much greater detail in subsequent chapters are discussed briefly, and attacks which will not be explored beyond this chapter are covered in greater detail. In much of the rest of this book we will focus on defenses against the attacks surveyed in this chapter.

A primary aim of an attack on machine learning, particularly deep learning, is to cause the neural network to make errors. Examples with severe implications include: fooling a biometric authentication system so that it grants access to sensitive material or building access to an unauthorized individual; fooling an automated breast cancer pre-screening system so that images with tumors are not forwarded to a radiologist; fooling an autonomous vehicle's recognition system so that it mistakes a stop sign for a speed limit sign. An attack may target the training dataset (its integrity or privacy, the former by data poisoning), the training process (deep learning), or the parameters of the deep neural network (DNN) once trained. Alternatively, or in addition, an attack may target vulnerabilities in the trained network by discovering test samples that produce erroneous output – such samples are called *adversarial inputs* or *test-time evasion attacks (TTEs)*. They have also been referred to as adversarial samples, adversarial examples, or (redundantly) adversarial attacks. All of these terms are ambiguous in light of backdoor triggers (Chapter 10) and querying/probing for purposes of reverse engineering (Chapter 14), which are also adversarial. Indeed, both TTEs (Chapter 4) and backdoor triggers produce incorrect outputs.

Defenses typically attempt to detect attacks and/or to proactively improve the robustness of machine learning in the face of them. They may also help to interpret the decision-making of a machine-learned system and to make it generally more robust even in the absence of an attack.

In this book, previously published attacks and defenses under various scenarios will be critically surveyed. The main focus is on unsupervised defenses with reasonable work factors against *strong* contemporary attacks on supervised machine-learned systems. Primarily, the examples are DNN classifiers applied to images, but there are some exceptions: Chapter 11 on 3D point cloud classifiers, Chapter 12 on non-image regression applications, and Chapter 13 on document classifiers.

In this chapter, after reviewing some jargon, an overview is first given of the three main types of attacks on machine learning that will be investigated in subsequent chapters.

1.1 Machine Learning and Its Attack Vectors

Machine learning involves learning predictive models (for tasks such as classification, regression, and time series prediction) from a finite training set of "examples." (Machine learning may also involve learning data "transformations," where one seeks the most informative/salient feature representation starting from high-dimensional feature vector examples that may involve many noisy/uninformative features.) Moreover, "deep learning" is simply machine learning applied to deep neural network (DNN) models – these involve numerous (in general nonlinear) layers of data processing applied to input patterns, culminating in the output of the DNN, which produces a classifier decision or a regression model prediction. Accordingly, attacks on machine learning/deep learning may target different stages of the machine learning model-building and use process: (i) corruption of the training data; (ii) malicious alteration of the learning process itself, or of the resulting model parameters; or (iii) disruption of the test-time use of machine learning models, so that they produce incorrect decisions/inferences. Attacks may also seek to reveal sensitive information, such as information about individual training examples (e.g., which patients participated in a large-scale medical study) or about a company's proprietary decision-making rule (e.g., for an investment bank, revealing how it makes its trading decisions).

The main attacks on machine learning that are comprehensively addressed in this book include: data poisoning attacks, backdoor attacks, test-time evasion attacks, membership-inference attacks, and reverse-engineering attacks. Data poisoning and backdoor attacks both involve corruption of the training set (and/or alteration of the learning process). However, backdoor attacks *also*, along with test-time evasion attacks, involve attacker exploits at test-time, that is, altering input patterns so as to produce erroneous model outputs. Membership-inference attacks seek to reveal sensitive information about training data, while reverse-engineering attacks aim to reveal a classifier's decision-making rule/a regression model's predictive rule, and the features on which it is based. All of these attacks (and defenses against them) will be studied in detail in this book.

1.2 Attacker/Defender Goals and Assumptions

Attack/defense scenarios typically begin by describing the attacker's specific goals and the knowledge and capabilities that the attacker and defender are assumed to possess. Also, attack/defense scenarios typically specify at what point the attacker/adversary and defender will act: before training, while the training dataset is being formed; during training; post-training but before operational deployment; at test/operational time; or during retraining or fine-tuning (including by "active" learning using judiciously chosen new supervised training data samples, or by "reinforcement" learning using recently observed test samples).

We now discuss the following post-training scenarios particularly germane to TTE attacks. In attack scenarios sometimes referred to as *black box*, the attacker does not

possess detailed knowledge of the trained DNN, but is assumed to be able to freely query the DNN, so as to learn its decision-making rule. Alternatively, in what is sometimes referred to as a *grey box* scenario, the adversary has access either to the DNN's parameters or to a training dataset (presumed i.i.d. with the training data used to build the targeted DNN) that allows it to build a good proxy of the DNN. In the following, "black box" will typically be used to describe *both* of those foregoing attack scenarios, since in both cases the attacker possesses (or creates) knowledge of the classifier, but not of any supplementary defense that may be mounted against the attack. In *white box* attack scenarios [32], the adversary has knowledge of the DNN parameters and is also assumed to have detailed knowledge of any mounted supplementary defense.[1] The latter could be obtained by an *insider* (inside attacker). Alternatively, if attack detection results in the classifier making a "rejection" or "undecided" decision, then the attacker may be able to learn the mechanism of the supplementary (anomaly detection) defense by observing this output (rejection decisions). [21] makes a case for white box attacks, ostensibly because "security through obscurity" may fail: the DNN parameters (as well as the inner details of a supplementary defense) may have been leaked to the attacker. It will be argued, however, that the privacy of a supplementary defense may be protected in some cases.

Let us next consider pre-training attack scenarios, germane to data poisoning attacks. If an attack targets the training set of a classifier, then one can imagine reasonable scenarios where the adversary can manipulate only a small fraction of the training samples (for example, to avoid detection). The adversary may either be unaware (black box) or aware (white box) of the remaining training samples.

It is important to assess the relative *work factors* of the attacker and defender, especially in the white box case. Work factors particularly include the computational effort and memory storage needed to mount a given attack or defense.

Extreme case scenarios, where either the attacker or defender is *omniscient* (knows everything) and/or *omnipotent* (can do anything) will not be considered here. For example, it may be theoretically impossible to defend against an adversary who is aware of and can easily manipulate all of the training dataset $\mathcal{X}$ and all aspects of the training process, that is, an omniscient and omnipotent insider. See, for example, Appendix R.4 of [286].

Note that calling a defense unsupervised (i.e., without any knowledge about a specific attack that may be mounted) is analogous to calling an attack black box (i.e., where the attacker has no knowledge of any supplementary defense). Similarly, calling a defense supervised is analogous to calling an attack white box. White box attacks and supervised defenses require information that generally may not be available in practice – they are arguably less realistic than black box attacks and unsupervised defenses.

Genuinely unsupervised defenses are also obviously preferable as they afford some protection against *zero day threats*, never before seen attacks, which are also sometimes

[1] This terminology is not standard. In some articles, knowledge of the model (its parameters and architecture) is dubbed "white box" or "grey box," and "black box" means that the adversary can only query the model. In the following, we will clarify these definitions when describing attack/defense scenarios.

called unknown unknowns. Supervised defenses, on the other hand, may only be effective against threats that are already known. Many proposed defenses appear to be unsupervised but either make unrealistic assumptions regarding limitations of the attack or involve hyperparameters which are difficult to set in an unsupervised, anomaly-detection setting.

Some attacks are *targeted*. For example, a targeted attack may focus on ensuring a DNN classifier assigns either a particular subset of data samples, for example, from a particular class (even a single sample), or a particular region of feature space, to the attacker's chosen (target) class [14]. Other attacks are *indiscriminate* (untargeted, or *error generic* [21]). An indiscriminate TTE or data poisoning attack on a classifier seeks to induce decision change *without* the requirement to misclassify to a particular class chosen by the attacker.[2]

Attacks may be referred to as *strong* when they both (i) succeed in their objective of inducing misclassifications and (ii) are not easily detectable by man or machine. For example, strong TTE attacks on a classifier craft examples that induce misclassifications to a target class but which to a human inspector still appear to be natural (artifact-free) examples from some other class. In fact, in general, a TTE attack is *only* deemed to be truly successful if it induces misclassifications while *not* introducing artifacts that make the attack either easily perceivable by a human being or easily detected by a trivial anomaly detector; that is, only strong attacks are truly successful ones.

1.3 Test-Time Evasion Attacks (TTEs) or Adversarial Inputs

As aforementioned, TTEs [19, 260] involve the alteration of test-time input patterns, resulting in erroneous decision-making/test-time inferences. TTEs are most commonly launched against statistical classifiers. However, they can also target other systems that involve discrete decision-making such as those that involve object detection and image segmentation [313]. From this standpoint, they can also in fact target unsupervised data clustering, where a learned clustering solution/model may be applied to identify to which cluster a new (test-time) data object should be assigned. TTEs are typically constructed with knowledge of the model/classifier, but without relying on knowledge of the specific training samples used for deep learning. Again, if the attacker has full knowledge of both the classifier and any deployed defense (and seeks to defeat both), it is referred to as a white box attack. How plausible the white box assumption is will be explored in Chapter 4.

Given a trained DNN, a TTE typically modifies a "clean" input sample in a way that may not be noticeable to a human inspector (in which case we refer to the modification as "imperceptible") but which induces a significant change in the DNN's output (decision).[3] We refer to the attacker's change to the input pattern as an adversarial

[2] Some authors use "universal" instead of "untargeted" and use "label-specific" instead of "targeted," in either a data poisoning or TTE context.

[3] In contrast, some "spoofing" attacks are not innocuous and the resulting change in the DNN's decision is not necessarily incorrect (not a misclassification).

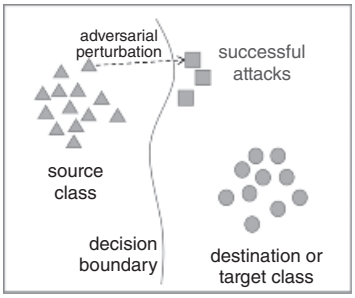


Figure 1.1 Illustration of a test-time evasion attack. Correctly classified source-class samples (triangles) are adversarially perturbed so that they are pushed across the decision boundary (represented by squares) and incorrectly classified to the destination/target class (the circle class). Reprinted from [179, 180] with permission.

perturbation of the input pattern. A strong, untargeted TTE modifies a clean, correctly classified sample $\underline{x}$ from class s, $\hat{c}(\underline{x}) = c(\underline{x}) = s$, so that the modified sample $\tilde{\underline{x}}$ is not classified to s, that is, so that $\hat{c}(\tilde{\underline{x}}) \neq s$ but while still appearing to be a *natural* (undoctored) class-s sample to a human inspector; see Figure 1.1. The imperceptibility objective is typically achieved by making the adversarial perturbation from $\underline{x}$ to $\tilde{\underline{x}}$ as small as possible according to some measure. Likewise, a targeted TTE attack seeks to induce decision change to a *target* class chosen by the attacker.

Clearly, in non-image domains that are high-dimensional (for which human beings either cannot intelligibly perceive class-discriminating patterns, or would require huge time and resources, perhaps involving data visualization tools, to make such inferences), it is much less of an imperative that the attack example be "imperceptible" to a human. However, for such domains, the attack should still aim to be evasive to simple, automated anomaly detection systems.

Note that TTE attacks in general assume the attacker knows the true class of the sample to be altered – otherwise, the attacker would only know he/she succeeded in inducing a decision change (to the actual classifier, or to the attacker's proxy) – *not* whether the change actually results in a misclassification (the decision change could otherwise in fact "correct" a misclassification). The practical implications of this assumption will be further explored in Chapter 4.

TTE attacks often employ techniques of neural network inversion (see Section 2.10) applied to an objective function whose maximization is consistent with a change in the classifier's decision (see Section 4.1). Some proposed TTE attacks compute the gradient of this objective but take a single, large step rather than taking an incremental descent approach involving smaller step sizes and (potentially numerous) small descent steps. As a result, to succeed in inducing misclassifications, the former approaches may require larger adversarial perturbations than the latter ones. However, some TTE attacks of the latter variety, with relatively small perturbations, can still be quite weak/ineffective.

As an example of the latter, let us consider a DNN classifier of handwritten digits trained on the MNIST dataset [141] with $K = 10$ classes, $\mathcal{Y} = \{0, 1, 2, \ldots, 9\}$.

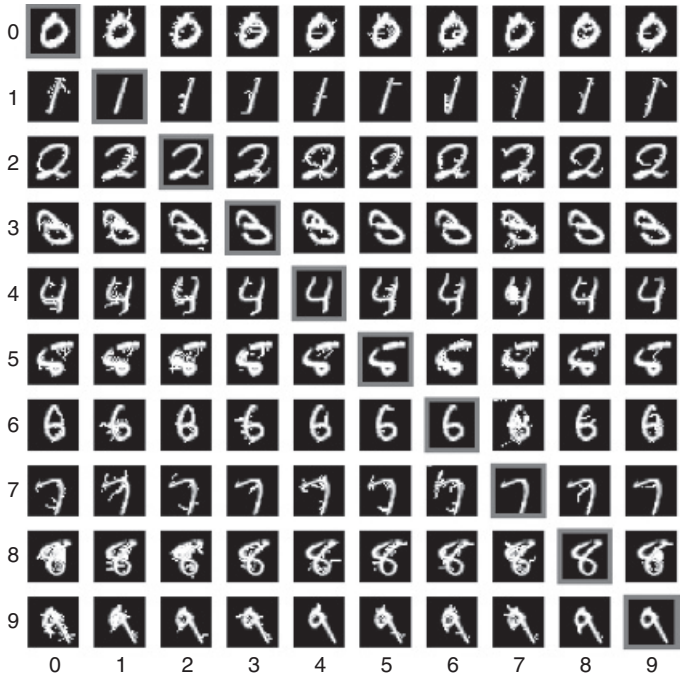


Figure 1.2 Images on the diagonal are examples of unmodified (clean) samples that are correctly classified by a DNN trained on MNIST [141]. Off-diagonal images are the result of the JSMA [203] attack applied to the diagonal image in the same row, which are classified by the DNN to the class corresponding to the column index. Adversarial examples are clearly doctored with significant salt and pepper noise. Reprinted from [179, 180] with permission.

Example JSMA TTE attacks (see Section 4.1), wherein individual pixels are modified sequentially until the desired misclassification is achieved, are depicted in Figure 1.2. Note the significant salt-and-pepper noise and extra white pixels introduced by the attack. A simple detection method based on the number of contiguous white regions in the image achieved 0.97 ROC AUC; see Table 6 of [179] and Section 4.10.2. Moreover, the attacked images should appear highly suspicious to a human observer. Thus, JSMA applied to MNIST is a weak TTE attack. Moreover, in some of the attack instances in Figure 1.2 even a human being cannot unambiguously assert the true class of origin for the image – note in particular many of the attacked '5's, some of the '3's, the attacked '6' classified as a '7', and the attacked '9' classified as a '0'. In this sense, the attack does not "fool" the classifier since even a human being cannot ambiguously determine the class of origin. Other TTE attacks such as FGSM [83] and CW [33] do exhibit grey ghosting artifacts on MNIST (see Figure 1.3) but these are less noticeable than JSMA artifacts. Moreover, all of these attacks exhibit much less noticeable artifacts for more complex image domains (involving textures and non-constant image background) such as CIFAR [129] and ImageNet [57].

A TTE attack may be on physical objects in the real world (e.g., altering a road sign or "camouflaging" a vehicle). However, in some cases it may be difficult to implement

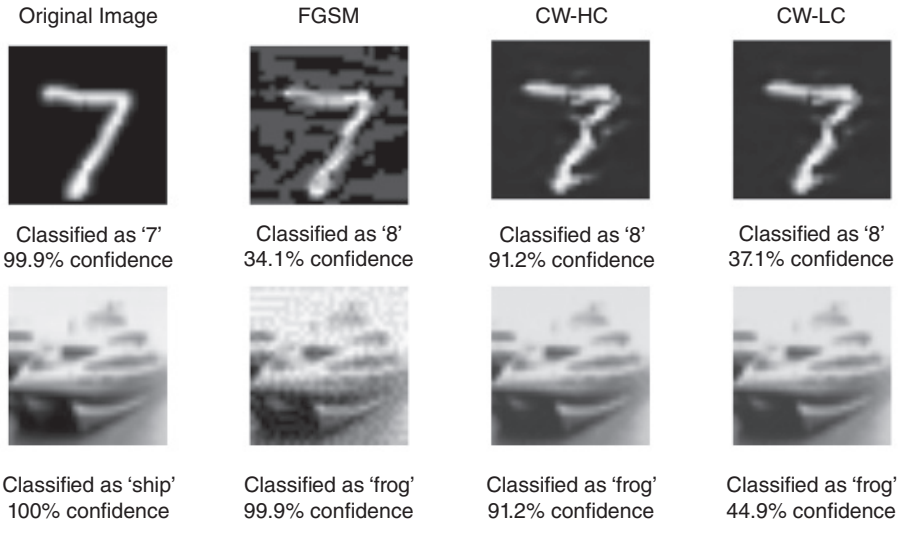


Figure 1.3 Examples of clean images, FGSM attack images [83], CW high confidence (CW-HC) images and CW low confidence (CW-LC) images [33], from MNIST [141] (first row) and CIFAR-10 [129] (second row) datasets. HC TTEs typically have much larger adversarial perturbations compared to LC TTEs. Reprinted from [284] with permission.

strong physical attacks. Alternatively, a TTE attack may alter data objects that have either already been digitally captured (e.g., digital images, voice files) or those which are *natively* digital (e.g., emails, documents, or computer programs). Such *digital* TTE attacks are invoked prior to the data object being input to the DNN.

While some attacks necessitate sophisticated defenses that may require significant innovations, other attacks may be defeated simply by invocation of standard techniques. For example, the use of encryption can defeat a man-in-the-middle TTE attack [135], that is, one wherein the data object is intercepted by the attacker, and modified, before being input to the classifier. As another example, the attack on voice systems in [31] could potentially be defeated by Apple Siri's existing built-in speaker-recognition system. Even if the speaker-recognition system can be overcome by existing voice cloning technology (especially under a white box scenario where the attacker has samples of the authorized party's voice), standard techniques of limited privilege can be applied. For example, Siri cannot enter data into the Safari web browser, and even if it could, there is standard multi-factor authentication to prevent unauthorized access to a private website even if the password were compromised.

Aside from use of standard security techniques, there are basically two different approaches to TTE defense. The first is robust classifier training, which seeks to be robust to (i.e., to correctly classify) adversarial inputs. The second, and more promising approach, is anomaly detection (AD) at test/operational time, which, unlike robust classification, makes explicit attack detection inferences. These two approaches are discussed in detail in Chapter 4.

1.4 Data Poisoning (DP) Attacks

1.4.1 Early Work on Error Generic DP attacks

Most early DP attacks simply seek to degrade the learned classifier's accuracy, see for example, [106, 176, 311]. For example, an early DP attack targeted spam/ham email classification and simply required the attacker to know common "good" words (in ham) and "bad" words (in spam) [14]. More recent "error generic" DP attacks require greater knowledge of the system under attack. [20] showed that significant degradation in the accuracy of a Support Vector Machine (SVM, see the Appendix) could be achieved with the addition of just one poisoned training sample – on MNIST the error rate increased from the 2–5% range to the 15–20% range. To achieve this, the attacker exploits knowledge of the training set, a validation set, the SVM learning algorithm, and its hyperparameters. The authors define the attacker's objective function as classifier error rate on the validation set as a function of the poisoned sample's location (and class label). This loss is maximized under the constraint that the support vector and non-support vector subsets of the training dataset are not altered by addition of the poisoned sample. Thus, [20] adds a single, new (adversarially labeled) support vector to the SVM solution. The constraint is met by performing gradient descent carefully, with a small step size. One would expect that even more classifier degradation could be achieved if the support vectors were allowed to *change* through the addition of the poisoned sample.[4] However, this would also entail a more complex optimization procedure. An illustrative example of such a DP attack on SVMs is shown in Figure 1.4.

While [20] required the attacker to possess substantial knowledge of the classifier, other works make even greater assumptions about an attacker's capabilities. In particular, in [176] it was noted that, in the data poisoning attack on active learning of an SVM in [175], the authors assumed that the *oracle* (typically a human expert) deliberately mislabels samples. Thus, the attack in [175] relies on even the human labeler being compromised.

While SVMs (which rely on a support vector subset of the training set to define the linear discriminant function) can unsurprisingly be fragile in the presence of DP attacks, there is little prior work investigating such attacks against DNNs. One reason may be computational complexity – one could in some way alternate gradient descent optimization in weight space (minimizing the loss function, i.e., the defender/learner's problem) and gradient *ascent* in pattern space (maximizing the loss function, i.e., the attacker's problem), to find a set of poisoned input patterns that maximally degrade the learned DNN's accuracy. However, such a procedure would be complicated and quite computationally heavy. Moreover, this assumes both that the attacker has access to the training set *and* that the attacker is the training authority.

DNNs should be less fragile in the presence of data poisoning than SVMs. To degrade a DNN's accuracy sufficiently, a larger fraction of poisoned samples may be

[4] On the other hand, if the attacker does *not* know the value of the margin slackness hyperparameter, he/she cannot ensure the poisoned sample will be a support vector; in such case, many more poisoned samples may be needed in order to significantly degrade SVM accuracy.

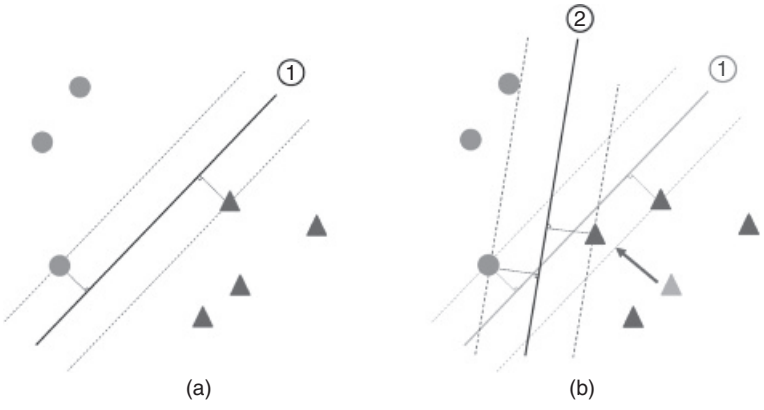


Figure 1.4 Linear SVM classifier decision boundary for a two-class dataset with support vectors and classification margins indicated (a). The decision boundary is significantly impacted in this example if just one training sample is changed, even when that sample's class label does not change (b). Here, the changed sample becomes a support vector. Reprinted from [180] with permission.

needed. For "big data" domains with, for example, one million training samples, even 2% data poisoning (with judicious selection of the poisoned samples by the attacker) means optimizing 20,000 poisoned sample locations and labels.

1.4.2 Backdoor DP Attacks on DNNs

On the other hand, DNNs appear to be quite vulnerable to *backdoor* DP attacks, as demonstrated in a number of works [44, 89, 150, 158, 268, 303, 304, 308]. Planted backdoors are also known as *Trojans*. To the training dataset, the attacker adds samples drawn from a source class with a backdoor pattern incorporated, and with the resulting poisoned samples labeled as belonging to a different (target) class. Thus, the classifier learns to classify to the attacker's target class whenever the attacker's *backdoor pattern* is embedded in a source-class test example to be classified, that is, when the test example is a backdoor trigger; see Figure 1.5. Successful backdoor poisoning does not significantly impact classification accuracy on clean (backdoor-free) test samples.

Triggering a backdoor of a (poisoned) DNN at test-time is typically much easier (and requires much less computation) compared to launching a TTE on a clean (unpoisoned) DNN with the same associated source and target classes.

The backdoor pattern could be an **imperceptible** (e.g., random-looking) local or global watermark-like pattern. Alternatively, it could be **perceptible but scene-plausible** – for example, the presence of glasses on a face [44], a plausible object in the background of an image scene (such as a tree or a bird in the sky, or a ball on a lawn), or a noise-like audio background pattern in the case of speech classification. Scene-plausible attacks on images may be implemented either by a physically "choreographed" scene or by photo-shopping a digital image to include the plausible object.

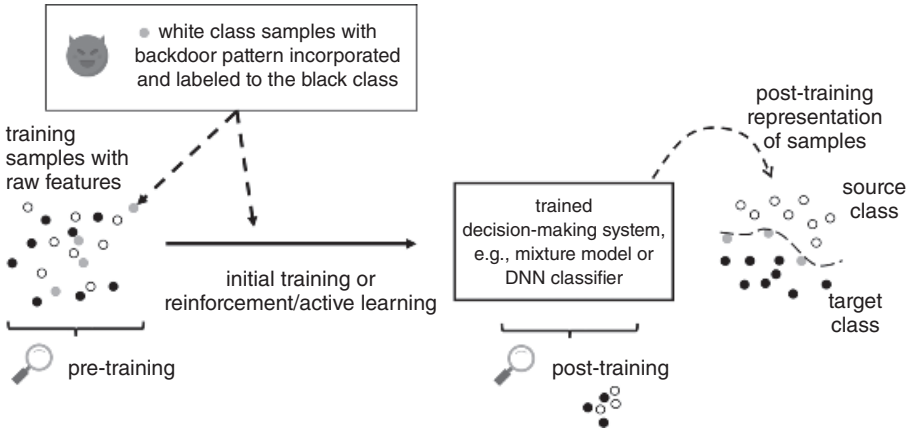


Figure 1.5 Illustration of a backdoor attack and the before/during and post-training defense scenarios. Also see Figure 6.1. The "in-flight" defense scenario, where backdoor triggers at test time are detected, is not depicted; see Chapter 10.

In Chapter 7, it will be argued that scene-plausibility implies that the backdoor pattern will be learned in a spatially invariant fashion by a DNN.

Various digital mechanisms can be applied to introduce a backdoor pattern into an image. These include replacing a local patch of pixels (consistent with the aforementioned perceptible, scene-plausible backdoors), applying an additive perturbation to the image, or applying a multiplicative perturbation. Noisy patch backdoor patterns are described in [144, 197]. [44, 316] employ a noisy patch $\underline{u}$ that is "blended" into a clean image $\underline{x}$ using an image-wide binary mask $\underline{m}$ (all elements are Boolean, $m_i \in \{0,1\}$ for all pixels i) and a real-valued blending factor $\alpha \in (0,1)$ to produce $\underline{\tilde{x}} = (\underline{1} - \alpha \underline{m}) \odot \underline{x} + \alpha \underline{m} \odot \underline{u}$. Here, $\odot$ is a pixel-wise product (e.g., a 3×3-pixel square mask in the same location for each poisoned image $\underline{\tilde{x}}$).

One very attractive aspect of backdoor attacks is that they may require *no* knowledge of the classifier – the attacker simply needs: (i) legitimate examples from the domain, into which it embeds the backdoor pattern; (ii) the ability to poison the training dataset with these samples labeled to the target (backdoor) class; and (iii) perhaps knowledge of the training set size, to know how many poisoned samples may be needed. On the other hand, if the attacker does possess knowledge of the classifier, its training set, and its learning algorithm, he/she can optimize the backdoor pattern to ensure: (i) the backdoor is well-learned; (ii) clean classifier accuracy is not compromised; and (iii) that (i) and (ii) are accomplished with the least amount of "attack strength," that is, with the fewest poisoned examples and/or using least noticeable backdoor patterns (so that the attack is not easily detected). An attempt at such an approach is given in [150].

Figure 1.6 is a low resolution picture of a car with a single pixel modification. A group of images with such changes could be used to poison a training set and thus plant a backdoor in the classifier. Note that, even at this level of resolution, it would be

Figure 1.6 Modified low resolution CIFAR-10 [129] image of a car with a single pixel changed to trigger (or train) a backdoor.

difficult for a human inspector to detect the poisoned samples in the training set or the samples triggering the backdoor operationally (at test time).

In some defenses, it is assumed that initially there is a clean (free of poisoned samples) training set, but that subsequently it is altered by additional (potentially unreliable) data collection, by online learning, or by the actions of an adversarial insider – under this scenario, the learner knows a subset of the available training set that is guaranteed to be *clean* (attack-free). In this case, for DP attacks, one can detect poisoned samples by discerning that their use in learning degrades classification accuracy (on a clean, held-out validation data subset) relative to just use of the clean subset of data [193].

A more challenging DP scenario for attack detection is the *embedded* scenario [301, 308], where one cannot assume the training dataset is "initially" clean and where there is no available means (using time stamps, data provenance, etc.) for identifying a subset of samples guaranteed to be free of poisoning.

Defenses against backdoor attacks under different scenarios are discussed in Chapters 5–10 for image classification, Chapter 11 for 3D point cloud classification, and Chapter 12 for regression.

Clean Label Backdoor Attacks

In [275], a *clean label* backdoor attack was proposed. These attacks have two key aspects. First, they intend to induce classification to the target class whenever the backdoor pattern is present, *irrespective* of the class of origin of the data object. Second, and most importantly, these attacks do not require any mislabeling of training examples. As motivation for the latter, suppose that the training set is initially *unlabeled* and that an honest human expert is responsible for labeling all the training samples.

In this situation, the mislabeling required for conventional backdoor attacks cannot be achieved. So, the attacker applies the backdoor pattern to samples that originate from the *target* class of the attack, not from a source class. Moreover, the poisoned samples may be further altered, for example, by:

- adding noise in order to weaken, within these samples, the features that can normally be relied upon to correctly classify them (but somehow in a way that is not noticed by the honest human expert labeling the samples); or
- adding a sample-specific adversarial perturbation, causing an unpoisoned classifier (trained on clean data and assumed available to the adversary) to change its class decision from the target class.

However, even when such sample alterations are applied, clean label backdoor attacks appear to require much greater attack strength than regular backdoor attacks in order to be successful. One reason is that some normal target-class features need to persist in the poisoned sample in order to convince the honest human labeler that the sample belongs to the target class. Alternatively, if (sample specific) adversarial perturbations are used, there is no reason why those perturbations will be learned instead of the target-class discriminating features when the victim classifier is trained. So, clean label backdoor attacks either require poisoning a much greater fraction of the training set than regular backdoor attacks or they require much more overt (and hence less evasive) backdoor patterns in order to overcome the target-class discriminating features in the poisoned samples. The former may not be possible in practice (as the attacker may have access to, or contribute to, only a portion of the training set). The latter will make the attack more easily detected (by either an automated detector or a human inspector). Consequently, clean label backdoor attacks do not appear to be a significant practical threat. See the results given in Figure 1.7 which involved an additively incorporated chessboard-watermark backdoor pattern, shown in Figure 6.4b, without any other alterations. Here, ten ResNet-18 (see Figure 2.6) classifiers were conventionally trained, each on a different CIFAR-10 subdomain of five randomly selected classes among which the poisoned target class was also randomly chosen. Note that the attack success rate is high only when the number of poisoned samples is large or when the perturbation size is quite large.

There are other types of backdoor attacks, e.g., label smoothed and sample specific.

1.4.3 Post-Training Model-Adjustment Attacks

Post-training, deployment-stage backdoor attacks have also been proposed, see for example, [12]. These could occur if the *model* is intercepted by an adversary (if it is transmitted from a remote site), that is, by a man-in-the-middle, or if the model is compromised by resident malware. These attacks are not discussed in detail here, but it is expected that they can be detected by the post-training backdoor defenses discussed in this book. Note also that a simple hash against the trained model parameters can be used to check if the model has been altered, post-training.

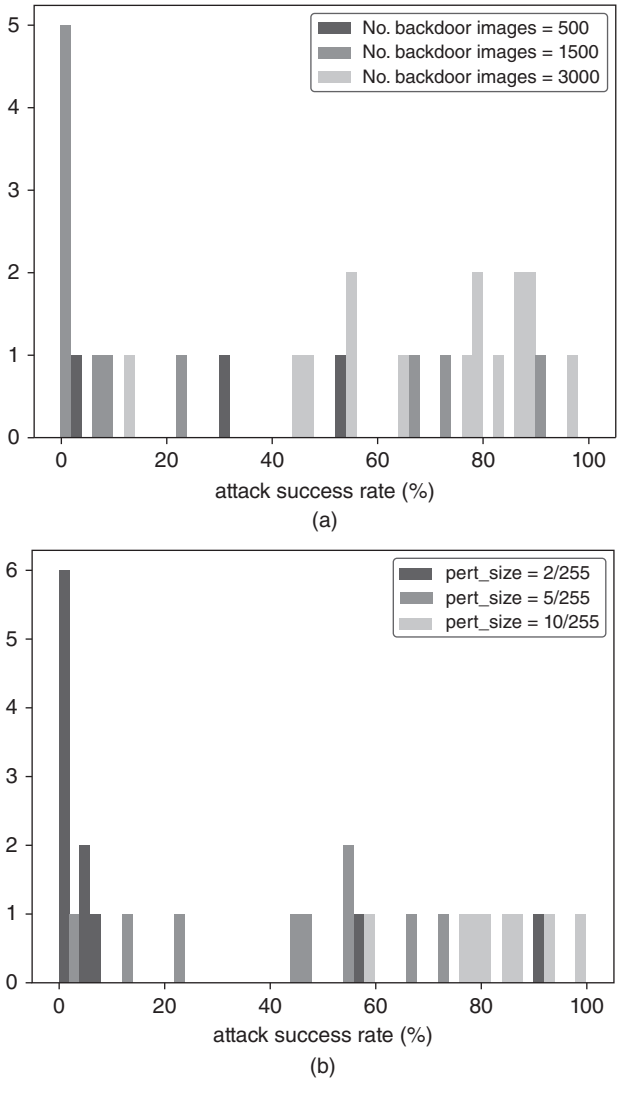


Figure 1.7 Histograms of attack success rate for clean label backdoor attacks, for the CIFAR-10 domain, with (a) different numbers of images for training set poisoning, and (b) different perturbation sizes for the backdoor pattern. To achieve a high attack success rate, either the poisoning rate must be very high, or the backdoor pattern size must be large (making the backdoor trigger potentially visually discernible).

1.5 Reverse-Engineering Attacks (REAs) Targeting the Deep Neural Network (DNN)

Let us now consider **privacy issues** related to reverse-engineering attacks (REAs) targeting the training set, the training process (algorithmic privacy), and/or the model

parameters and architecture. The identification of some training samples or of some aspects of the trained model by an adversary is sometimes called "data leakage."

REAs may involve querying (probing) a DNN numerous times, either to learn its decision rule or to learn something about the dataset on which it was trained (an attack on data privacy). Repeated querying can be used to create a training set for the attacker (with the classifier's decision on each example used as the supervising label), allowing him/her to learn a surrogate of the true classifier. Several motivations have been given for reverse engineering a classifier's decision rule.

In [267], the authors consider black box machine learning (ML) services, offered by a company, where, for a given domain, a user (with limited resources for learning a model) pays for class decisions on individual samples (queries) submitted to the ML service. [267] demonstrates that, with a significant number of queries (e.g., tens of thousands, even for low-dimensional classification domains), one can learn a classifier that closely mimics black box ML service decisions. Once the black box has been reverse engineered, the attacker need no longer subscribe to the ML service. Perhaps more importantly, such reverse engineering enables TTE attacks by providing the attacker with knowledge of the classifier when it is not initially known. One weakness of [267] is that it considers neither very large (feature space) domains nor very large neural networks – for orders of magnitude more queries may be needed to reverse engineer a DNN on a large-scale domain. However, a more critical weakness of [267], discussed further in Chapter 14, is that its queries should be easily detected because they are *random*, that is, they do not use any knowledge of the nominal (training) data distribution for the given domain.

REAs based on more realistic queries have been proposed. In [202], the adversary collects a small set of representative labeled samples from the domain as an initial training set and uses this to train an initial surrogate classifier. Then, there is data collection and retraining over a sequence of stages. In each, the adversary augments the current training set by querying the classifier with the stage's newly generated samples. Each successive stage crafts samples closer to the classifier's true decision boundaries (see (14.1)), which is helpful for surrogate classifier learning (but which also makes these samples less class representative and thus more detectable). Once a sufficiently accurate surrogate classifier is learned, a TTE attack can be launched using it. Defenses against REAs are discussed in Chapter 14.

1.6 Attacks on Privacy of Training Data

Another emergent attack, referred to as a membership-inference attack, seeks not to alter classifier decision-making but rather to glean, from the classifier, (assumed sensitive) information about the training set on which it was learned. Relevant applications here include: discerning whether a particular person participated in a patient study that produced a disease classifier (or a diagnostic or prognostic decision-making aid) – one may then infer he/she is likely to possess the disease; or discerning whether a particular individual's data was used in training a system that grants secure access (to

a building, data, financial records) only to company employees or vetted individuals. In the latter case, one might alternatively seek to infer what such individuals *look like* (estimate an image of an employee's face [71]). There are various scenarios we can consider for this type of problem.

An important scenario is one wherein the attacker only has black box (query) access to the classifier. A representative approach that investigates data privacy attacks on classifiers under this scenario is [244]. This work makes some strong assumptions, but shows that when these assumptions hold one can make quite accurate inferences of whether or not an individual's sample was used in training (a membership-inference attack), for example, accuracies as surprisingly high as 80–90% in inferring training-set membership on two classification domains. The authors pose the attacker's problem as learning a posterior model whose input is a data record (feature vector) and whose output is the probability that the data record was used in training the classifier under attack. There are three pivotal assumptions made.

- The attacker has query access to the victim classifier and, when queried, the classifier does not merely produce decisions – it gives decision "confidence" that could consist of the vector of posterior probabilities over all classes, just the "top" probabilities, or quantized values for these probabilities. The attacker does not query the victim classifier repeatedly to reverse engineer its function – this could be detected using the methods of Chapter 14. It simply queries using the data sample on which it is seeking to violate privacy, and elicits the victim classifier's decision and confidence on this sample.
- It is assumed that the attacker has access to a (surrogate) dataset that is statistically similar to the training set used in building the victim classifier [244]. This assumption is plausible only in some applications. In the patient study scenario, the attacker could have access to data records from hospital B, while the study yielding the victim classifier was produced by hospital A – hospital B's population could be very similar to that of A. However, it is less plausible that the attacker would have a training set statistically similar to a particular company's dataset, used to build its secure authentication classifier – unless, for example, members of this company also belong, in large numbers, to the same country club.
- Even though the victim classifier *is* assumed to be a black box with respect to the attacker, it is assumed that the victim classifier was trained using a particular online tool or "ML pay-for-service" system (e.g., provided by Google), that the attacker knows which tool/service was used, and he/she also has access to this tool. In this way, even without knowing what the classifier type is (e.g., SVM or a particular DNN architecture), the attacker can assume that, *given a similar dataset, with the same feature vector format, from the same classification domain*, the tool/service is likely to produce a new classifier (a shadow classifier) that "behaves in a similar way" as the classifier under attack. In particular, it should exhibit similar decision confidence patterns for samples used for training (high decision confidence) compared to samples not used for training.

Based on its surrogate training (and test) datasets, the attacker uses the tool/service to build an array of such shadow models. For each such model, he/she produces class decisions and the confidence scores on the surrogate training samples, and separately on the test samples. Each such triple class decision, confidence vector, training set example (Yes/No) is treated as a supervised instance of a *new* training set, used to learn a binary posterior model that infers whether a given sample was part of a shadow model's training set. After the attacker's binary classifier is trained, it can be applied to the output of the classifier under attack, when queried by a given data sample, and yield the probability that the query sample was part of the attacked classifier's training set.

It was noted there are strong assumptions in this work, whose violation could substantially minimize the amount of membership "leakage" obtained. First, if the victim classifier does not produce decision confidence, but merely a decision, this would defeat this attack. Producing confidence on decisions is important in order for classifier decisions to be trusted, but one could, for example, grossly coarsen the victim classifier's output confidence to "highly confident," "confident," "weakly confident," "uncertain" – such quantization could potentially defeat the membership-inference attack of [244] .

Second, as the authors note, this attack is successful because trained classifiers tend to overfit to training examples, producing patterns of "high confidence" on samples used for training, and patterns of lower confidence for non-training samples. The authors investigate defense strategies that seek to reduce classifier overfitting, or at least its signature in the victim classifier's posterior. These mitigations include use of regularization and altering the victim classifier's posterior to increase its decision entropy. While strong regularization can degrade the accuracy of the attacker's binary (training set example: Yes/No) classifier, it may also compromise accuracy of the victim classifier.

One can also likely defeat or weaken this attack by *not* using an accessible (and inferrable) service for training the victim classifier. This attack may not transfer well if the victim and shadow classifier decision rules are quite different.

One can also simply suppose that after training the targeted classifier, the training set that was used is (securely) retained by the platform/system that operates the classifier [180]. Now, when the classifier is queried by a sample, the system can first check whether the sample is part of the training dataset. If the sample is not part of the training set, the classifier can output its decision and confidence, as usual. However, if the query sample is in the training set (or is even essentially indistinguishable from a training pattern, i.e., if the attacker added a small amount of noise to the query sample in order to be evasive), the system can infer that this is very likely a data privacy attack query. In this case, the classifier should still output the correct decision, as well as confidence values that are at least maximum a posteriori (MAP) consistent with that decision. However, the system should randomize the confidence values to destroy any privacy revelation, and thus confound the attacker. This simple defense should