

Jinyin Chen · Ximin Zhang ·
Haibin Zheng

Attacks, Defenses and Testing for Deep Learning



Springer

Attacks, Defenses and Testing for Deep Learning

Jinyin Chen · Ximin Zhang · Haibin Zheng

Attacks, Defenses and Testing for Deep Learning



Springer

Jinyin Chen
Institute of Cyberspace Security, College
of Information Engineering
Zhejiang University of Technology
Hangzhou, Zhejiang, China

Ximin Zhang
College of Information Engineering
Zhejiang University of Technology
Hangzhou, Zhejiang, China

Haibin Zheng
Institute of Cyberspace Security, College
of Information Engineering
Zhejiang University of Technology
Hangzhou, Zhejiang, China

ISBN 978-981-97-0424-8 ISBN 978-981-97-0425-5 (eBook)
<https://doi.org/10.1007/978-981-97-0425-5>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature
Singapore Pte Ltd. 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721,
Singapore

Paper in this product is recyclable.

Preface

Deep learning, as a powerful machine learning technique, has made tremendous breakthroughs and advancements in various fields in recent years. By mimicking the structure and function of human brain neural networks, it can extract complex feature representations from massive data and achieve efficient pattern recognition and decision-making. With the improvement of computing power and the rapid growth of data, deep learning has become the core technology in fields such as computer vision, natural language processing, and speech recognition, and has shown extensive application prospects in practical scenarios.

The core idea of deep learning is to achieve complex pattern recognition and decision-making tasks by constructing deep neural networks. These deep neural networks consist of multiple layers of neurons, where each layer performs some non-linear transformations and feature extraction on the input data. Through the forward propagation process, the network can generate output results based on the learned parameters. At the same time, using the backpropagation algorithm, the network can adjust the parameters according to the differences with the true results, thus continuously optimizing the performance of the model.

However, deep learning also faces challenges and issues. Deep learning is vulnerable to security threats such as adversarial attacks and poisoning attacks, which can lead to unpredictable consequences. For example, in 2018 alone, there were 12 autonomous driving accidents worldwide involving companies like Uber, Tesla, Ford, and Google that are developing AI for autonomous driving. Therefore, in this era of data-driven computation powered by deep learning algorithms, it is urgent to propose more advanced attack algorithms to uncover potential vulnerabilities in deep neural networks. Additionally, measures need to be taken to strengthen the defense mechanisms of deep neural networks against security vulnerabilities. Finally, it is essential to test and evaluate deep neural networks to ensure their safety and robustness.

The book aims to provide a comprehensive introduction to the methods of attacks, defenses, and testing evaluations for deep learning in various scenarios. We focus on multiple application scenarios such as computer vision, federated learning, graph neural networks, and reinforcement learning, considering multiple security issues that exist under different data modalities, model structures, and tasks. Through the

study of practical case studies on attack and defense security applications, we hope to provide new ideas and methods for the further development of deep learning.

The book is divided into three main parts: attacks, defenses, and testing. In the attack section, we introduce in detail the attack methods and techniques targeting deep learning models. These attacks aim to exploit vulnerabilities in the model to compromise its performance or privacy, including adversarial attacks, poisoning attacks, and backdoor attacks. For each attack type, we describe its motivations, methods, and attack effects in detail to help readers better understand and respond to these threats.

Attacks on deep neural networks can be classified into adversarial attacks and poisoning attacks based on the attack stage. Adversarial attacks occur during the model testing phase, where an attacker adds carefully designed tiny perturbations to the original data to generate adversarial samples, thereby deceiving the deep learning model and causing it to misclassify with high confidence as a malicious attack. Poisoning attacks occur during the model training phase, where an attacker injects poisoned samples into the training dataset, thus embedding a backdoor trigger in the trained deep learning model. When a poisoned sample is input during the testing phase, the attack is triggered.

In Chap. 1, Chen, Su, Zhang, and Shen propose a black-box adversarial attack method based on genetic algorithms. They generate initial perturbations by randomly generating and using the classic white-box adversarial attack method AM. By combining genetic algorithms, they design a fitness function to evaluate and constrain sample individuals from both attack capability and perturbation control aspects. This allows them to compute approximate optimal adversarial samples.

In Chap. 2, Chen, Zhang, Zheng, and Ming introduce a novel adversarial network consisting of one generator and two discriminators to address this problem. The generator automatically extracts hidden features of the target class and embeds them into benign training samples. One discriminator controls the poisoning rate of perturbations, while the other acts as the target model to demonstrate the poisoning effect.

In Chap. 3, Chen, Zheng, and Xiong present a Generative Adversarial Network (GAN) for poisoning attacks, aiming to solve the problem that poisoned samples are easily detected by defense algorithms. By using Gradient-weighted Class Activation Mapping (Grad-CAM) to calculate channel-spatial attention and pixel-spatial attention, they can focus on the target contour and generate smaller perturbations with better attack performance and less disturbance.

In Chap. 4, Chen, Huang, Zheng, Yu, Jiang, and Cui introduce a method to steal global node embeddings and establish a shadow model for the attack generator on the server side. They then add noise to the node embeddings to confuse the shadow model and generate attacks by using gradients of pairs of nodes guided by the noisy node embeddings.

In Chap. 5, Chen, Huang, Zheng, Jin, and Chen present a method that locates key neurons using critical neuron paths and modifies the features of important nodes. They build a shadow model of the server model by manipulating data and the probability returned by the server model, generating adversarial samples.

In Chap. 6, Chen, Huang, Zheng, and Lin introduce an adversarial attack method that estimates the classification boundaries of different classes. This method calculates the minimum perturbation matrix for the target classification boundary to achieve misclassification of the attacked nodes.

In Chap. 7, Chen, Xiong, and Zhang present a backdoor attack framework. The framework generates multiple initial triggers using a Generative Adversarial Network (GAN) and then selects some links from the initial triggers based on the gradient information of the attack discriminator in the GAN to form a set of triggers. This reduces the size of the triggers and improves the stealthiness of the attack.

In Chap. 8, Chen and Wang introduce an adversarial attack based on attention mechanisms, utilizing potential vulnerabilities in deep reinforcement learning (DRL). They construct target action agent manipulators and target state agent manipulators to generate stealthy poisoned examples with the agent manipulators and fine-tune the model together with clean examples.

Effective defense methods are important guarantees for the application of deep learning in security-sensitive areas. Existing defense methods can be divided into three categories based on their defensive nature: data modification defense methods, model modification defense methods, and plug-in detector defense methods. Data modification defense methods use fine-tuning of input data to filter out adversarial perturbations and achieve adversarial defense. Model modification defense methods start from within the model and adjust neurons and model frameworks to achieve adversarial defense. Plug-in detector defense methods train adversarial sample detectors to prevent adversarial samples from being input into deep learning models, achieving adversarial defense.

In Chap. 9, Chen, Jin, Chen, Zheng, and Xuan introduce a plug-in detector defense method. By computing the local gradients of benign examples and misclassified noisy examples, the detector is trained to accurately distinguish adversarial examples and even naturally misclassified inputs from benign examples.

In Chap. 10, Chen, Zheng, and Chen present a defense method that extracts image features using pixel channel attention and pixel plane attention, filtering out adversarial perturbations. This method addresses the problem of divergence in visualization heatmap areas and blurred object contours leading to model misclassification. The state-of-the-art defense methods have been shown to improve when cascaded with this method.

In Chap. 11, Chen, Jin, Chen, Zheng, and Yu introduce a defense method against unknown attacks. They calculate the influence of neurons in selected layers based on several batches of benign samples and then identify potential front neurons and tail neurons that may be exploited by opponents. Next, they strengthen the front neurons and suppress the tail neurons to generate counteracting perturbations that can mitigate and offset general adversarial perturbations.

In Chap. 12, Chen, Zheng, Shan, Wen, and Ji introduce a lightweight detector based on cuckoo search. They use cuckoo search to select the approximate optimal type of channel transformation and the minimum number of channel transformations to improve efficiency in sample rotation and resizing.

In Chap. 13, Chen, Li, Liu, and Zheng present a model modification defense method based on weight evolution frequency. This method defines the concept of Weight Evolution Frequency Matrix (WEF-Matrix) to record the weight evolution frequency of the penultimate layer of the model. Based on this, it calculates the change in weights between two consecutive local training rounds and uses the average value of the total range of changes as a dynamic threshold to evaluate the evolution frequency of all weights. This prevents freeloaders from obtaining a global model with model weights contributed only by benign clients and stealing well-trained high-quality models.

In Chap. 14, Chen, Li, Zheng, and Cheng introduce a model fingerprint generation method for server models by using key samples. The server extracts global model features and generates a set of adversarial examples as model fingerprints. A detector is trained using the feature distribution of key samples to predict the true class encoding of key samples. When testing suspicious models, the feature distribution of key samples output by the suspicious model is obtained, and the detector is used to predict the true label of the key samples.

In Chap. 15, Chen, Ma, Li, Liu, and Liu present a data modification defense method in the field of federated learning. This method proposes a fake label generator to obtain uniformly distributed labels and then designs a noise generator to protect sensitive attributes. The experimental results show that this method achieves state-of-the-art performance while balancing privacy protection and task accuracy.

In Chap. 16, Liu, Li, Zheng, Ming, and Chen introduce an adversarial attack method by collecting clean small datasets used for main task training in federated learning on the server side to generate adversarial examples. They observe the behavior of updated models under adversarial examples, use clustering algorithms to select benign models and eliminate other models, and use the clustering results for backdoor model detection to achieve defensive effects.

Testing and evaluating deep neural networks is an effective way to measure the security and robustness of deep learning models. Through testing and evaluation, potential security vulnerabilities and weaknesses in deep neural networks can be identified. These vulnerabilities may be caused by issues such as overfitting, data bias, or incomplete training data. By identifying and fixing these vulnerabilities, the security and robustness of the model can be improved. In addition, testing and evaluation can verify the performance of deep neural networks in various scenarios. By extensively testing the model, its applicability and accuracy in the real world can be determined. This helps ensure the reliability and effectiveness of the model in practical applications.

In Chap. 17, Jin, Chen, Zheng, Wang, Xiong, and Ming introduce a model-agnostic robustness evaluation metric based on feature distribution, which addresses the time-consuming nature of current robustness evaluations of deep models and their dependence on specific attacks and model structures. This method includes two aspects: feature subspace aggregation within the same class and feature subspace distance between different classes.

In Chap. 18, Zheng, Jin, and Chen present a testing method that simplifies the problem of measuring improper behavior probability into measuring the difficulty

of movement in the feature space. This method provides a formal guarantee for the lower bound of the movement cost and calculates the movement cost value based on GEVT to complete the testing evaluation.

In Chap. 19, Zheng, Chen, Du, Zhang, Wang, Cheng, Ji, and Chen introduce a testing method based on AS curves and AUC measurement, addressing the issue that the effectiveness of testing evaluations is affected by gradient vanishing. This method qualitatively and quantitatively explains the severity of discrimination at each layer of DNN, systematically searches the input space, and performs testing evaluations on the model.

In Chap. 20, Chen, Zhang, Xu, Fu, Zhang, Zhang, and Xuan introduce an end-to-end dynamic link prediction model for deep learning. This model can automatically learn structural and temporal features of networks within a unified framework and predict links that have never appeared in the network. Extensive experiments show that this model significantly outperforms newly proposed methods for dynamic network link prediction and achieves state-of-the-art results.

Finally, during the writing process of the book, we received a lot of help from many people. Here, we would like to express our special thanks to Qi Xuan, Zhaoxia Shi, Tao Liu, Haonan Ma, Minying Ma, Zhiqi Cao, Xinran Liu, and Chengyu Jia for their selfless support. Without their help and encouragement, this book would not have been completed on time.

This research was supported by the NSFC (No. 62072406), Zhejiang Provincial Natural Science Foundation (No. LDQ23F020001).

Hangzhou, China
November 2023

Jinyin Chen
Ximin Zhang
Haibin Zheng

Contents

Part I Attacks for Deep Learning

1 Perturbation-Optimized Black-Box Adversarial Attacks via Genetic Algorithm	5
1.1 Introduction	5
1.2 Related Works	7
1.2.1 White-Box Adversarial Attack Methods	7
1.2.2 Black-Box Adversarial Attack Methods	9
1.3 Methodology	9
1.3.1 Problem Definition	9
1.3.2 Framework	10
1.3.3 Initialization	12
1.3.4 Fitness Function	12
1.3.5 Evolutionary Operations	13
1.3.6 Generation Update	15
1.4 Evaluation Metrics	15
1.4.1 Perturbations Evaluation Metric	15
1.4.2 Attack Evaluation Metrics	17
1.5 Experiments	18
1.5.1 Experiment Setup	18
1.5.2 Baseline Methods	18
1.5.3 Attack Performance Comparison	19
1.5.4 Influence of Initialization Strategy	20
1.6 Conclusion	22
References	22
2 Feature Transfer-Based Stealthy Poisoning Attack for DNNs	25
2.1 Introduction	25
2.2 Methodology	27
2.2.1 Main Framework Architecture	27
2.2.2 The Model Loss	27

2.3	Experiments and Analysis	28
2.3.1	Setup	28
2.3.2	The Effectiveness of Our Method	29
2.3.3	The Stealthiness of Our Method	32
2.4	Conclusion	33
	References	33
3	Adversarial Attacks on GNN-Based Vertical Federated Learning	35
3.1	Introduction	35
3.2	Related Work	36
3.2.1	Inference Attack on Federated Learning	37
3.2.2	Adversarial Attacks on GNN Models	37
3.3	Methodology	38
3.3.1	Problem Definition	38
3.3.2	Threat Model	40
3.3.3	Our Method	41
3.3.4	Complexity Analysis	46
3.4	Experiments	46
3.4.1	Datasets	46
3.4.2	Local GNN Model	47
3.4.3	Attack Baselines	48
3.4.4	Experiment Setup	48
3.4.5	Attack Performance	48
3.5	Conclusion	52
	References	53
4	A Novel DNN Object Contour Attack on Image Recognition	55
4.1	Introduction	55
4.2	Related Works	56
4.3	Methodology	57
4.3.1	Perturbation Distribution of Adversarial Attack	57
4.3.2	Spatial Attention	59
4.4	Experiments	61
4.4.1	Setup	62
4.4.2	Metrics	62
4.4.3	Performance of Our Method for White-Box Attack	63
4.4.4	Performance of Our Method for Black-Box Attack	64
4.4.5	Visualization Analysis	66
4.4.6	Confidence Analysis	67
4.4.7	Attack Comparison Against Defense	71
4.5	Conclusion	72
	References	72

5 Query-Efficient Adversarial Attack Against Vertical Federated Graph Learning	75
5.1 Introduction	75
5.2 Related Work	77
5.3 Preliminary and Problem Formulation	78
5.3.1 Vertical Federated Graph Learning	78
5.3.2 Adversarial Attack on Vertical Federated Graph Learning	78
5.3.3 Threat Model	79
5.4 Methodology	80
5.4.1 Data Manipulation	81
5.4.2 Shadow Model Construction	82
5.4.3 Adversarial Attacks	83
5.5 Experiments	84
5.5.1 Experimental Settings	84
5.5.2 Effectiveness and Efficiency on Attacking VFGL	87
5.6 Conclusion	93
References	95
6 Targeted Label Adversarial Attack on Graph Embedding	97
6.1 Introduction	97
6.2 Related Work	99
6.2.1 Graph Embedding Methods	99
6.2.2 Graph Attacks	100
6.3 Preliminaries	101
6.3.1 GCN Model	101
6.4 Methodology	102
6.4.1 Main Framework	103
6.4.2 Adversarial Graph Generation	104
6.4.3 Targeted Label Attack	107
6.4.4 Transfer Adversarial Attack	108
6.5 Experimental Results	108
6.5.1 Experimental Setup	108
6.5.2 Attack Performance	110
6.6 Conclusion	116
References	116
7 Backdoor Attack on Dynamic Link Prediction	119
7.1 Introduction	119
7.2 Related Work	121
7.2.1 Dynamic Link Prediction	121
7.2.2 Backdoor Attacks on GNNs	121
7.2.3 Adversarial Attacks on DLP	122
7.3 Preliminary	122
7.3.1 Problem Definition	122
7.3.2 Threat Model	123

7.4	Methodology	124
7.4.1	Trigger Generator	125
7.4.2	Trigger Gradient Exploration	126
7.4.3	Optimization of GAN	127
7.4.4	Filter Discriminator	128
7.4.5	Backdoored Model Implementation	128
7.4.6	Theoretical Analysis	129
7.5	Experiments and Discussion	130
7.5.1	Datasets	131
7.5.2	Baseline Methods	131
7.5.3	Metrics	132
7.5.4	Experiment Setup	133
7.5.5	Overall Performance of Backdoor Attacks	133
7.5.6	Attack Transferability	138
7.5.7	Trigger Injection Timestamp Analysis	139
7.6	Conclusion	140
	References	140
8	Attention Mechanism-Based Adversarial Attack Against DRL	143
8.1	Introduction	143
8.2	Related Works	144
8.3	Preliminaries	146
8.3.1	The Basic Model of RL	146
8.3.2	Attention Mechanism	146
8.3.3	DRL Feature Transformation	147
8.4	Methodology	147
8.4.1	General Introduction to Attack	147
8.4.2	Attack Method Description	149
8.5	Experimental Evaluation	151
8.5.1	Experimental Description	151
8.5.2	Experimental Results	151
8.6	Conclusion	155
	References	155

Part II Defenses for Deep Learning

9	Detecting Adversarial Examples via Local Gradient Checking	159
9.1	Introduction	159
9.2	Related Work	161
9.2.1	Adversarial Attacks	161
9.2.2	Adversarial Detections	161
9.3	Methodology	162
9.3.1	Overview	162
9.3.2	Calculation of Local Gradient	162

9.3.3	Detection via AdvD	163
9.3.4	Complexity Analysis	165
9.4	Experiments and Analysis	165
9.4.1	Experiment Setup	165
9.4.2	Detection Against Adversarial Attacks	167
9.4.3	Detection Against Misclassified Natural Inputs	169
9.5	Conclusion	170
	References	170
10	A Novel Adversarial Defense by Refocusing on Critical Areas and Strengthening Object Contours	173
10.1	Introduction	173
10.2	Related Works	175
10.2.1	Defense for DNNs	175
10.2.2	Attention Mechanism for DNNs	176
10.3	Methodology	176
10.3.1	Feature Map Extraction	176
10.3.2	Pixel Channel Attention	177
10.3.3	Pixel Plane Attention	178
10.4	Experiments	179
10.4.1	Setup	179
10.4.2	Defense Against White-Box Attack	182
10.4.3	Defense Against Black-Box Attack	182
10.4.4	Visualization in Cascade Process	191
10.5	Conclusions	193
	References	193
11	Neuron-Level Inverse Perturbation Against Adversarial Attacks	197
11.1	Introduction	197
11.2	Related Work	199
11.2.1	Defenses Against Attacks	199
11.2.2	Neuron-Level Approaches for DNN's Security	200
11.3	Preliminaries	201
11.3.1	Threat Model	201
11.3.2	Definitions	201
11.4	Methodology	202
11.4.1	Framework	202
11.4.2	Neuron Selection Module	203
11.4.3	Generation of Neuron Template	203
11.4.4	Construction of Candidates	203
11.4.5	Adaptivity and Reclassification	204
11.4.6	Algorithm Complexity	205
11.5	Experimental Settings	206
11.5.1	Datasets	206
11.5.2	Models	206

11.5.3	Attacks	206
11.5.4	Defense Baselines	206
11.5.5	Evaluation Metrics	208
11.6	Evaluation and Analysis	208
11.6.1	RQ1: Effectiveness	209
11.6.2	RQ2: Generality	212
11.6.3	RQ3: Efficiency	213
11.7	Conclusion	214
	References	214
12	Adaptive Channel Transformation-Based Detector for Adversarial Attacks	217
12.1	Introduction	217
12.2	Related Works	219
12.2.1	Detection Defenses	219
12.3	Methodology	220
12.3.1	Framework	220
12.3.2	Channel Transformation Candidates	221
12.3.3	ACT Based on BCS	222
12.3.4	Detector Structure and Training Strategies	225
12.4	Experiments and Analysis	226
12.4.1	Setup	226
12.4.2	Visual Analysis of Channel Transformations	226
12.4.3	Comparison of Detection Results	230
12.5	Conclusions	234
	References	234
13	Defense Against Free-Rider Attack from the Weight Evolving Frequency	237
13.1	Introduction	237
13.2	Related Work	238
13.2.1	Free-Rider Attacks on Federated Learning	239
13.2.2	Defenses Against Free-Rider Attacks	239
13.3	Methodology	240
13.3.1	Overview	240
13.3.2	WEF-Matrix Information Collection	240
13.3.3	Client Separation	242
13.3.4	Personalized Model Aggregation	243
13.3.5	Algorithm Complexity	244
13.4	Experiments Setting	244
13.5	Evaluation and Analysis	245
13.5.1	RQ1: Defense Effectiveness of Our Method	246
13.5.2	RQ2: Defensive Effects at Higher Free-Riders Ratios	246
13.5.3	RQ3: Trade-Off Between Defense and Main Task Performance	249

13.6 Conclusion	252
References	253
14 An Effective Model Copyright Protection for Federated Learning	255
14.1 Introduction	255
14.2 Related Works	257
14.2.1 Centralized Model IP Protection	257
14.2.2 IP Protection in FL	257
14.3 Preliminaries and Background	257
14.3.1 Horizontal Federated Learning	258
14.3.2 Key Samples	258
14.3.3 Model Fingerprints	258
14.4 Threat Model	259
14.5 Methodology	260
14.5.1 Model Fingerprints Generation	260
14.5.2 Model Fingerprinting Adaptive Enhancement	263
14.5.3 Detector Training	264
14.5.4 IP Verification	265
14.5.5 Algorithm Complexity	266
14.6 Experiments Design and Setup	266
14.7 Evaluation and Analysis	268
14.7.1 RQ1: Validity	268
14.7.2 RQ2: Fidelity	269
14.7.3 RQ3: Robustness	269
14.8 Conclusion	274
References	275
15 Guard the Vertical Federated Graph Learning from Property Inference Attack	277
15.1 Introduction	277
15.2 Related Work	279
15.2.1 Inference Attack on VFL	279
15.2.2 Privacy-Preserving Methods	279
15.3 Methodology	280
15.3.1 Details of Strategies	281
15.3.2 Theoretical Discussion	282
15.4 Experiments	283
15.4.1 Datasets	284
15.4.2 Models	284
15.4.3 Defense Baselines	284
15.4.4 Metrics	285
15.4.5 Loss Functions	286
15.4.6 Experiment Setup	286
15.4.7 RQ1: Attack Effectiveness	287
15.4.8 RQ2: Defense Effectiveness	288

15.4.9	RQ3: Sensitivity Evaluation	290
15.5	Conclusions	293
	References	293
16	Using Adversarial Examples to against Backdoor Attack in Federated Learning	297
16.1	Introduction	297
16.2	Background and Related Work	298
16.3	Threat Model	299
16.3.1	Attacker's Goal	299
16.3.2	Attacker's Capability	299
16.3.3	Defender's Knowledge and Capability	300
16.4	Methodology	300
16.4.1	Model Detection	301
16.4.2	Model Aggregation	302
16.5	Experiments	303
16.5.1	Datasets and Models	304
16.5.2	Experiment Setup	304
16.5.3	Effectiveness of Our Method	305
16.5.4	Interpretability of Our Method via Neuron Activation	305
16.5.5	Our Method Against Adaptive Attacks	308
16.6	Conclusion	309
	References	309

Part III Testing for Deep Learning

17	Evaluating the Adversarial Robustness of Deep Model by Decision Boundaries	315
17.1	Introduction	315
17.2	Related Works	317
17.2.1	Robustness Evaluation of Deep Models	317
17.2.2	Decision Boundary and Adversarial Samples	318
17.2.3	Robustness Based on Adversarial Attack and Defense	318
17.3	Methodology	319
17.3.1	Preliminaries	319
17.3.2	The Proposed Metrics for Robustness Evaluation	320
17.3.3	The Key Properties of Our Method	321
17.4	Experiments	322
17.4.1	Setup	322
17.4.2	Benchmark Robustness Measured by Our Metric on Deep Models	325
17.4.3	Evaluation of Robustness on Defensive Models	327
17.5	Conclusion	329
	References	329

18 Certifiable Prioritization for Deep Neural Networks via Movement Cost in Feature Space	333
18.1 Introduction	333
18.2 Related Works	335
18.3 Background	335
18.3.1 Deep Neural Networks	336
18.3.2 Definitions of Inverse Perturbation	336
18.4 Methodology	337
18.4.1 A Movement View in Feature Space	337
18.4.2 Formal Guarantees for Movement Cost	338
18.4.3 Movement Cost Estimation via GEVT	340
18.4.4 Prioritization Through Movement Cost	341
18.5 Experimental Setting	341
18.5.1 Subjects	341
18.5.2 Baselines	343
18.5.3 Measurements	343
18.5.4 Implementation Details	345
18.6 Experimental Results and Analysis	345
18.6.1 Effectiveness (RQ1)	345
18.6.2 Efficiency (RQ2)	349
18.6.3 Robustness (RQ3)	350
18.6.4 Generalizability (RQ4)	352
18.7 Conclusions	352
References	354
19 Interpretable White-Box Fairness Testing Through Biased Neuron Identification	357
19.1 Introduction	357
19.2 Related Works	359
19.3 Background	360
19.4 Methodology	361
19.4.1 Quantitative Discrimination Interpretation	362
19.4.2 Interpretation-Based IDI Generation	364
19.4.3 Generalization Framework on Unstructured Data	366
19.5 Experimental Setting	368
19.5.1 Datasets	368
19.5.2 Classifiers	368
19.5.3 Baselines	369
19.5.4 Evaluation Metrics	369
19.6 Experimental Results	371
19.6.1 Research Questions 1	371
19.6.2 Research Questions 2	379
19.7 Conclusions	379
References	380

20 A Deep Learning Framework for Dynamic Network Link Prediction	383
20.1 Introduction	383
20.2 Related Work	384
20.3 Methodology	385
20.3.1 Problem Definition	385
20.3.2 Our Framework	386
20.3.3 Balanced Training Process	389
20.3.4 Complexity Analysis	392
20.4 Experiments	392
20.4.1 Datasets	392
20.4.2 Baseline Methods	393
20.4.3 Evaluation Metrics	394
20.4.4 Experimental Results	396
20.5 Conclusion	397
References	398

Part I

Attacks for Deep Learning

Attacks on deep neural networks can be classified into adversarial attacks and poisoning attacks based on the attack stage. Adversarial attacks occur during the model testing phase, where an attacker creates adversarial samples by adding carefully designed tiny perturbations to the original data, thus fooling the deep learning model into making incorrect judgments with high confidence. Poisoning attacks occur during the model training phase, where an attacker injects poisoned samples into the training dataset, embedding a backdoor trigger in the trained deep learning model. When a poisoned sample is input during the testing phase, the attack is triggered.

Currently, there are several issues with attacks on deep neural networks: (1) Most black-box adversarial attack algorithms fail to achieve the desired success rate; (2) existing adversarial attack methods mainly focus on the success rate of patch-based sample attacks, making poisoned samples easily detectable by defense algorithms; (3) updating perturbations based on gradient information ignores the role of feature extraction in deep learning models; (4) GVFL faces vulnerabilities in practical applications due to distrust; (5) vertical federated graph learning (VFGL) has gained rapid development but lacks research on adversarial attacks; (6) most attack methods struggle to accurately misclassify instances to the target label; (7) DLP algorithm has loopholes in data collection and training; (8) lack of transferability.

Chapter 1 introduces a black-box adversarial attack method based on genetic algorithms to solve the problem of unsatisfactory success rate in black-box adversarial attacks. This method generates initial perturbations by randomly generating and using the classic white-box adversarial attack method AM. It combines genetic algorithms, designs fitness functions, and evaluates and constrains sample individuals from both attack capability and perturbation control aspects. By calculating, it obtains approximate optimal adversarial samples, solving the problem that most black-box adversarial attack algorithms cannot achieve the expected success rate compared to white-box attacks. Experimental results show that this method outperforms existing black-box attack methods in terms of attack capability and perturbation control.

Chapter 2 introduces a Generative Adversarial Network (GAN) for poisoning attacks to solve the problem of poisoned samples being easily detected by defense algorithms. This network consists of a feature extractor, a generator network, and a discriminator network. Under the GAN framework, the generator minimizes the loss between the pixels of poisoned samples and benign samples to achieve stealthiness by binding the size of perturbations. The discriminator evaluates the similarity between poisoned samples and original samples. The feature extractor minimizes the loss between the features of poisoned samples and their own features to improve attack success rate. Experiments show that this network has higher attack success rates on multiple datasets than five comparison attack algorithms and better stealthiness.

Chapter 3 introduces a white-box targeted attack to solve the problem of ignoring the role of feature extraction in deep learning models. This adversarial attack method uses Gradient-weighted Class Activation Mapping (Grad-CAM) to calculate channel-space attention and pixel-space attention. Channel-space attention reduces the attention area of deep neural networks, while pixel-space attention achieves error localization of target contours. By combining the two, it can focus features on target contours to generate smaller perturbations and produce more attackable adversarial samples with less disturbance.

Chapter 4 introduces a new GNN vertical federated learning attack method to address the vulnerability of GVFL in practical applications due to distrust crises. Firstly, it steals global node embeddings and establishes a shadow model for the attack generator on the server side. Secondly, noise is added to node embeddings to confuse the shadow model. Finally, an attack is generated by leveraging gradients between pairs of nodes under the guidance of noisy node embeddings.

Chapter 5 presents a query-efficient adversarial attack framework for studying adversarial attacks in VFGL. Key neuron paths are located to identify key neurons, and corresponding important node features are modified to increase the contribution of malicious clients. A shadow model of the server model is established by manipulating data and the probability returned by the server model.

Chapter 6 introduces a novel target label attack on graph embeddings to solve the problem that most attack methods struggle to accurately misclassify instances to the target label. First, it estimates the classification boundaries for different categories. Then, it calculates the minimum perturbation matrix based on the target classification boundary to misclassify the attacked nodes. Finally, it modifies the adjacency matrix according to the maximum absolute value of the perturbation matrix. This method achieves state-of-the-art attack performance with minimal perturbations.

Chapter 7 presents a novel backdoor attack to address the vulnerability of DLP algorithms in data collection and training. It generates multiple initial triggers using a Generative Adversarial Network (GAN) and selects partial links from the generated triggers based on the gradient information of the attack discriminator in the GAN to form a trigger set, thereby reducing the size of the trigger and improving the stealthiness of the attack. This method successfully launches a backdoor attack on state-of-the-art DLP models with a success rate exceeding 90%.

Chapter 8 introduces an adversarial attack based on attention mechanisms to address the lack of transferability of adversarial samples. It generates more effective adversarial samples by fully utilizing the hidden features extracted through attention operations in DRL. Both channel attention and pixel attention are applied to extract features, modifying the clean state to an adversarial state.

Chapter 1

Perturbation-Optimized Black-Box Adversarial Attacks via Genetic Algorithm



1.1 Introduction

Deep learning is a fundamental aspect of modern machine learning and artificial intelligence [1]. However, recent research has revealed a significant vulnerability of deep learning models—their susceptibility to adversarial examples. Adversarial examples are created by adding small perturbations to original images, which can lead deep models to produce incorrect predictions [2–6]. This phenomenon was first introduced by Szegedy et al. [7] and has since gained considerable attention as a prominent research topic. The existence of adversarial attacks poses a significant threat to the reliability and trustworthiness of deep model-based applications. For instance, facial recognition-based payment systems can be easily fooled by adversarial glasses, allowing individuals to impersonate others. Similarly, autonomous driving systems relying on image recognition become perilous if road signs can be carefully crafted as adversarial examples. Therefore, it is crucial to comprehend the mechanisms behind these attacks and develop effective defense strategies. Numerous adversarial attack methods have been proposed to study the vulnerabilities of deep learning models and enhance their defensibility. Examples include the Jacobian-based Saliency Map Attack (JSMA) [8], DeepFool [9], One-pixel Attack [10], and Limited Queries and Information Attack [11]. These methods contribute to a better understanding of adversarial attacks and aid in strengthening model defenses.

Adversarial attacks can be broadly categorized into three types: gradient-based attacks, score-based attacks, and transfer-based attacks [12]. Gradient-based attacks, also known as white-box attacks, leverage detailed model information to craft adversarial examples. Examples of gradient-based attacks include Basic Iterative Method (BIM) [13], Houdini [14], and DeepFool [9], which exploit gradient loss during the attack process. On the other hand, score-based attacks, also referred to as oracle attacks, do not require access to internal model information and instead make predictions based on query access to the model. Black-box attacks represent another category, where attacks are conducted without knowledge of the target model's internal configuration. These attacks can utilize transfer learning across models [15] or

require access to all training datasets [16]. In practical scenarios, most real-world systems do not disclose the weight, architecture, or training data of their models, making white-box attacks and equivalent model attacks challenging to execute. However, there is still a need for effective black-box attacks that do not rely on internal model information. To address this challenge, we have developed a completely internal model information-independent adversarial attack. This attack not only fills the gap in evolutionary-based adversarial attacks with minimal perturbation, but also facilitates the evaluation and enhancement of the defensibility of current state-of-the-art deep models.

In real-world scenarios, attackers often have limited information about the target model, such as the top few predicted classes and their corresponding confidence scores. In black-box attacks, the objective is to reduce the confidence of the true label with minimal perturbation. This can be seen as an optimization problem, where the goal is to find the optimal perturbation that achieves the desired outcome. Genetic algorithms are widely employed as effective optimization tools in various applications, such as energy optimization [17], distribution network optimization [18], ontology alignments optimization [19], and web crawling [20], consistently delivering excellent optimization performance. In this chapter, we propose a novel adversarial perturbation optimization attack based on a genetic algorithm to conduct black-box attacks. We construct a fitness function that combines classification confidence and perturbation size, aiming to find the approximate optimal adversarial example. Additionally, we design genetic operations that ensure the generation of approximate optimal perturbations. By leveraging the genetic algorithm, our attack approach enables effective black-box attacks by finding the perturbations that undermine the confidence of the true label while minimizing their magnitude.

Existing adversarial attack methods commonly utilize different perturbation evaluation metrics, such as L_0 , L_2 , and L_∞ , based on the characteristics of the algorithm being used [21]. For instance, L-BFGS [7] and FGSM [22] employ the L_∞ metric, JSMA [8] and one-pixel [10] use L_0 , while DeepFool [9] adopts L_2 . Consequently, it becomes challenging to compare different perturbations and algorithms, as they are evaluated from different perspectives. To address this limitation, we propose a novel perturbation assessment method based on the sensitivity of the human visual system. This new method allows for a comprehensive evaluation of perturbations, making them more closely aligned with actual sensitivity. By considering the characteristics of human perception, we aim to provide a more holistic and accurate assessment of the quality of perturbations, enabling a fair comparison among different algorithms.

The main contribution of our work can be concluded as

- **Perturbation Optimization.** Our method introduces a novel approach for perturbation optimization, enabling the generation of effective black-box adversarial examples. It achieves a high success rate in attacking various deep learning models while providing control over the magnitude of perturbations.
- **White-box Comparable Black-box Attack.** Our method optimizes perturbations by leveraging the confidence of the black-box model's output. In many cases, it achieves a high attack success rate that is comparable to white-box attack methods.

- **New Perturbations Evaluation Metric.** We propose a novel perturbation evaluation metric that provides a comprehensive assessment of the perturbation. This metric considers different dimensions and maps the size of the perturbation to each dimension, resulting in an evaluation result that more realistically reflects the impact of the perturbation.
- **Improve Defense Capability through Adversarial Training.** We utilize adversarial examples generated by our method to train deep models and enhance their defense capabilities. Through experiments, we demonstrate that adversarial training using examples from our method outperforms other attack methods in terms of improving model defense.

1.2 Related Works

In this section, we provide an overview of classic adversarial attack methods. The concept of adversarial attacks against deep models was initially introduced by Szegedy [7], which led to the development of numerous adversarial attack techniques [9, 22–25]. Furthermore, researchers have extended adversarial attacks to various applications, including speech recognition systems [26], malware detectors [27, 28], and face recognition systems [29]. Adversarial attacks can be broadly categorized into two types: white-box attacks and black-box attacks. White-box attacks have knowledge about the internal structure of the target model, whereas black-box attacks lack this information.

1.2.1 White-Box Adversarial Attack Methods

A white-box attack involves exploiting knowledge of the internal structure of a target model to launch an attack. It is the most commonly used attack method and can also be adapted to achieve black-box attacks by attacking equivalent models. Currently, researchers have proposed numerous white-box attack methods [8, 9, 13, 22, 23]. For instance, Bose et al. introduced adversarial attacks on face detectors using constrained optimization based on neural networks [30]. Yu et al. presented a fast framework for generating adversarial examples using adversarial saliency prediction [31]. Chen et al. developed a robust physical adversarial attack on the faster R-CNN object detector [32]. Ramanathan et al. proposed adversarial attacks on computer vision algorithms using natural perturbations [3].

In this chapter, the perturbation optimization algorithm leverages adversarial examples generated by white-box attacks as partial initial solutions and employs a genetic algorithm to optimize the perturbations. To ensure diversity in the initial solutions, we select seven different attack methods for generating them. The rationale for choosing these methods is explained in detail in Sect. 1.3. The following section provides a comprehensive description of the attack methods.

Fast Gradient Sign Model (FGSM) [22] FGSM is a popular and simple non-targeted adversarial attack method. It utilizes the gradients obtained through back-propagation from the target deep neural network (DNN) to generate adversarial examples. Perturbation is evaluated by $\rho = \epsilon sign(\nabla J(\theta, I_c, I))$ [1], where ∇J denotes the gradient of the original image I_c around the model parameters θ . $sign(.)$ denotes the sign function, and ϵ is a small scalar value that limits the disturbance norm.

Basic Iterative Model (BIM) [13] BIM, also known as Projected Gradient Descent, is a widely used adversarial attack method. It is an extension of the single-step method, where multiple small-step iterations are performed with adjustments in the direction after each step. By iterating over a sufficient number of steps, BIM can successfully generate an adversarial example that is classified as the target label. BIM is considered a standard convex optimization method in the field of adversarial attacks.

Jacobian-Based Saliency Map Attack (JSMA) [8] JSMA is an adversarial attack method that constructs a saliency map based on the input–output relationship of the target deep neural network (DNN). It iteratively modifies the most important pixels in the input image according to the saliency map in order to deceive the network. During each iteration, JSMA recalculates the saliency map and uses the derivative of the DNN with respect to the input image as a modification index for the adversarial attack. This greedy search process continues until either the number of modified pixels reaches a predefined threshold or the deception is successful. JSMA leverages the saliency information to efficiently and effectively generate adversarial examples.

DeepFool [9] DeepFool is a simple yet highly effective untargeted adversarial attack method. In each iteration, it calculates the minimum distance, denoted as $d(y_1, y_0)$, required for each label y_1 to cross the decision boundary by approximating the model’s decision boundary with a linear hyperplane. Here, y_1 represents the label of the adversarial example classified by the deep model, and y_0 represents the true label of the adversarial example. DeepFool then takes appropriate steps in the direction of the closest class boundary. The perturbations to the image are accumulated over each iteration, and the final perturbation is computed once the output criteria changes. DeepFool is known for its effectiveness in generating adversarial examples by finding the shortest distance to the decision boundaries of different classes.

Carlini and Wagner Attacks (C&W) [23] The Carlini and Wagner Attack is considered one of the most powerful adversarial attacks. It is characterized as a sophisticated iterative gradient attack that incorporates the Adam optimizer. The attack leverages the internal architecture and configuration of the target deep neural network (DNN) to guide the adversarial attack. Additionally, the C&W Attack utilizes the L_2 norm to measure the dissimilarity between the adversarial image and the original image. By combining these techniques, the C&W Attack achieves remarkable effectiveness in generating adversarial examples.

Gaussian Blur [33] Gaussian Blur is a linear smoothing filter commonly employed in image processing to reduce Gaussian noise. It is widely used in the noise reduction process. Essentially, Gaussian filtering involves the weighted averaging of the entire image. Each pixel’s value is obtained by taking a weighted average of its own value

and the values of neighboring pixels. However, while Gaussian filtering effectively reduces noise, it can also cause the image to lose certain details, leading to potential errors in CNN classification.

Salt and Pepper Noise [34] Salt and pepper noise, also known as impulsive noise, introduces random variations to pixel values in an image. This type of noise manifests in binary images by randomly turning some pixels black (pepper) or white (salt). The presence of salt and pepper noise can significantly impact the classification accuracy of Convolutional Neural Networks (CNNs), as these noisy pixels may lead to misclassifications. The occurrence of salt and pepper noise in an image increases the likelihood of misclassifying the corresponding CNN label due to the altered pixel values.

1.2.2 *Black-Box Adversarial Attack Methods*

When dealing with a black-box model, the attacker only has access to the input and output of the target model, while the internal configuration remains inaccessible. In the case of image classification using Convolutional Neural Networks (CNNs), an image is provided as input, and the model produces confidence scores for each possible label as output [35]. In practical scenarios, many target models are black-box models. Therefore, black-box attacks have garnered significant research attention and have been explored by various researchers [36–38]. For instance, Milton et al. [38] proposed the Momentum Diverse Input Iterative Fast Gradient Sign Method (M-DI2-FGSM) to attack black-box facial recognition systems. Dong et al. [37] introduced a range of momentum-based iterative algorithms to enhance adversarial attacks. Brendel et al. [12] presented the Boundary Attack, which opened new avenues for studying the robustness of machine learning models and raised concerns about the safety of deployed systems. Ilyas et al. [11] developed black-box adversarial attacks with limited information queries. Currently, the state-of-the-art black-box attack methods are ZOO and Boundary, both aiming to enhance the success rate of the attacks.

1.3 Methodology

1.3.1 *Problem Definition*

In the context of image recognition, let's consider an example denoted as S . This example is passed through a Deep Neural Model (DNN), which assigns a label to S based on its classification. To launch an attack on the target model TM , an Attack Method (AM) is employed to generate a perturbation denoted as A . This perturbation is then added to S to create an adversarial example referred to as AS . The objective is

to generate AS in such a way that it causes the target model TM to output an incorrect label, effectively fooling the model. It is important to note that the perturbation A added to S should be negligible, ensuring that AS closely resembles the original image.

Definition 1.1 (*DNN-Based Image Label*). Deep Neural Network (DNN) is trained by a large number of labeled images. For a given image S , DNN can output label y_1 , represented as $TM(\Theta, S) = y_1$, where Θ represents parameters, y_1 means the output label of the highest confidence.

Definition 1.2 (*Adversarial Attack*). Given a DNN for S , whose response output is $TM(\Theta, S) = y_0$, Attack Method AM generates an adversarial image AS to make $TM(\Theta, AS) = y_1$, and $y_0 \neq y_1$, where S and AS are almost indistinguishable.

Adversarial attacks are initiated by utilizing adversarial examples, which are typically created by introducing perturbations to the original image. The effectiveness of an attack heavily relies on the quality of the perturbation added. In the case of black-box attacks, the objective is to either select the most optimal adversarial example generated by white-box attacks or to generate an adversarial example that surpasses the capabilities of existing white-box attacks. This concept is illustrated in Fig. 1.1. The generation of a black-box adversarial example can be viewed as an optimization problem, where the aim is to find the optimal perturbation that maximizes the attack success.

1.3.2 Framework

In this study, we introduce a novel approach called Perturbation Optimization Black-box Attack based on Genetic Algorithm (referred to as our method). Our method utilizes the initial perturbations of adversarial examples and optimizes them using a fitness function denoted as $\phi(\cdot)$ to obtain an approximate optimal adversarial example AS_{opt} . The framework of our proposed method is outlined in Fig. 1.2.

Figure 1.2 depicts the process of generating a high-quality adversarial example using the genetic algorithm. Initially, we employ two strategies to generate initial perturbations for evolution: random perturbations and perturbations generated by classic white-box adversarial attack methods (AM). These perturbations are added to the original example S to create the corresponding initial adversarial examples $AS^{t=0}$, where $t = 0$ denotes the first generation of the population. Next, we define a fitness function $\phi(AS^t i)$ to evaluate the t^{th} iteration example $AS^t i \in AS^t$, where i ranges from 0 to n , and t represents the t^{th} iteration generation. This fitness function allows us to assess the quality of each adversarial example. Then, we employ typical genetic algorithm operators, including selection, crossover, and mutation, to evolve a new generation for perturbation optimization. In each iteration generation (t^{th} iteration), two adversarial examples $AS^t i$ and $AS^t j$ are selected as parents, and crossover and mutation operations are performed to generate children $AS^{t+1} i$ and $AS^{t+1} j$. This

evolutionary process continues until a termination condition is satisfied. By utilizing the genetic algorithm as an efficient optimization tool, we can generate perturbations

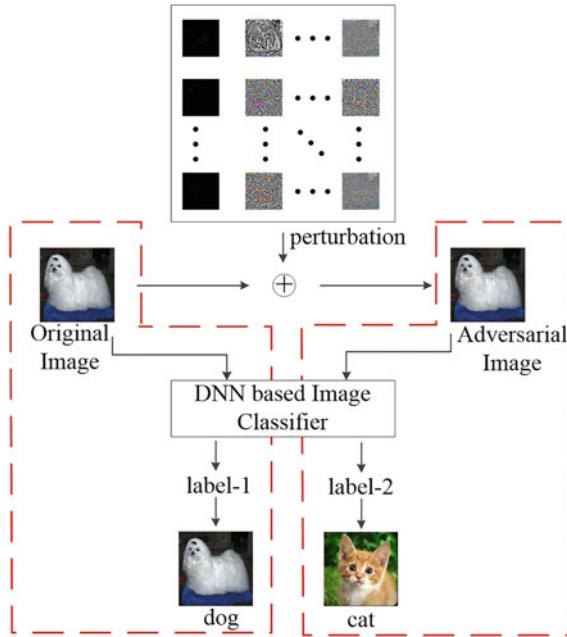


Fig. 1.1 An illustration of black-box attack. Black-box attacks can be viewed as choosing the best countermeasure example generated from a white-box attack or generating a better confrontation example than the current white-box attack

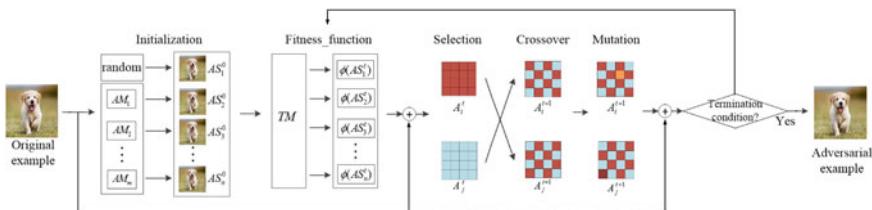


Fig. 1.2 Algorithm block diagram of ours. Perturbations are regarded as solutions to achieve approximate optimal attack performance. Ours is established in stages: (1) Initialization, perturbations are generated from different attack methods. (2) Calculate fitness, adversarial examples are classified by Target Model to obtain their output label and confidence. Fitness function is designed to assess attack capacity and perturbation size. (3) Perturbation is collected by the difference between adversarial example and original one, and optimized by evolutionary operations (selection, crossover, and mutation). (4) Iterations are carried on until approximate optimal perturbation is achieved. The approximate optimal adversarial example will be output

for black-box adversarial examples, thereby enhancing the quality of the generated adversarial examples.

1.3.3 Initialization

The initialization phase plays a critical role in generating the initial solutions at the start of the algorithm. The quality of these initial solutions has a direct impact on the subsequent iterations. If the initial solutions are close to the approximate optimal solution, the algorithm will converge more rapidly. However, if the initial solutions are far from the optimal solution, the algorithm will require more iterations to converge towards the global optimum. Apart from focusing on the quality of the initial solutions, it is also important to consider the diversity of these solutions. Studies have shown that maintaining diversity in the initial population of a genetic algorithm can help ensure the exploration of a wider search space and improve the chances of reaching an approximate global optimum [39].

To enhance the diversity of the initial solutions, this chapter presents two methods for generating initial examples.

Random Perturbation Initialization The objective of the optimization process is to evolve an approximate optimal adversarial example AS_{opt} from a given example S . In the random perturbation initialization method, a search distribution of random Gaussian noise is applied around the original image S [11]. This can be represented as $AS^{i=0} = S + \sigma\delta$, where $\delta \sim \mathcal{N}(0, 1)$ denotes a random variable following a standard normal distribution, and σ represents the search variance.

Adversarial Perturbation Initialization To ensure diversity within the population, the initial adversarial examples are generated using various white-box attack methods (AM). In this study, we employ several attack methods, including FGSM, BIM, JSMA, DeepFool, and C&W, as discussed in Sect. 1.2. These attack methods are applied to three different target models (VGG16, VGG19, and ResNet50) to generate adversarial examples. As a result, a total of 21 attack examples AS^0i are generated. Among them, FGSM and BIM limit the perturbation using the L_∞ norm, which ensures that the change in each pixel falls within a certain range. JSMA restricts the perturbation using the L_0 norm, which limits the number of pixels that can be modified within a certain range. DeepFool limits the perturbation using the L_2 norm, in addition to the L_∞ norm. C&W, on the other hand, restricts the perturbation using the L_0 , L_2 , and L_∞ norms, providing comprehensive restrictions on perturbations from multiple aspects.

1.3.4 Fitness Function

The fitness function is defined to assess the quality of examples in the genetic algorithm (GA). The choice of an appropriate fitness function is crucial, as it directly

impacts the convergence speed of the genetic algorithm and determines whether it can successfully find the optimal solution.

Excellent adversarial examples typically exhibit two main characteristics. Firstly, they closely resemble the original image, with only minimal perturbations that are imperceptible to the human eye. Secondly, these adversarial examples tend to be confidently misclassified by the target model. Therefore, the design of the fitness function should consider the confidence of misclassification and the size of perturbations. This can be represented by Eq. 1.1.

$$\phi(AS_i^t) = P(AS_i^t) - \alpha Z(A_i^t) \quad \alpha \in [0, 1]. \quad (1.1)$$

In the equation, $\phi(AS_i^t)$ represents the fitness function for the example AS_i^t . The attack performance of AS_i^t is denoted by $P(AS_i^t)$, which is calculated using Eq. 1.2. The term $Z(A_i^t)$ refers to the size of the perturbation in the adversarial example, calculated using Eq. 1.7, where $A_i + S = AS_i$. This paper introduces a novel perturbation metric called $Z(A_i^t)$, which will be explained and compared in Sect. 1.4. Additionally, the coefficient α is used to adjust the proportion of attack performance and perturbation size. For instance, when $\alpha = 0$, the fitness function only takes into account the attack performance $P(AS_i^t)$. On the other hand, a larger value of α leads the optimization process to prioritize the size of the perturbation. In such cases, the optimized adversarial example may have a relatively lower attack performance but with a significantly smaller perturbation.

$$P(AS_i^t) = \begin{cases} p(y_1|AS_i^t) - p(y_0|AS_i^t) & y_1 \neq y_0 \\ p(y_2|AS_i^t) - p(y_0|AS_i^t) & y_1 = y_0, \end{cases} \quad (1.2)$$

where y_0 means the true label of the adversarial image, and y_1, y_2 indicate top one and two output confidence of the target model for AS_i^t . The term $p(y|AS_i^t)$ represents the confidence that the adversarial example AS_i^t is classified as label y by the target model. A successful attack occurs when the output label differs from the true label of AS_i^t . In this case, the attack performance is measured by the confidence gap between the top predicted label y_1 and the true label y_0 . A larger confidence gap indicates a stronger attack capability, as it signifies a greater deviation from the correct classification. Conversely, if the output label is the same as the true label, it indicates a failed attack. In such cases, we calculate the confidence gap between the top predicted label (true label) and the second highest predicted label. The larger the confidence gap, the more challenging it is for the attack to be successful.

1.3.5 Evolutionary Operations

1.3.5.1 Selection Operator

The selection operator is used to choose two parent examples from the population, which will then produce two children through crossover and mutation operators.

Generally, examples with higher fitness values are more likely to be selected as parents. In this paper, we adopt the Roulette wheel selection method [40]. For a given example AS^t_i , its selection probability can be calculated using Eq. 1.3. The interval $(fr(AS^t_i - 1), fr(AS^t_i)]$ is then determined for each model using Eq. 1.4. This interval represents the range within which the example's selection probability falls.

$$f(AS^t_i) = \frac{\phi(AS^t_i)}{\sum_{j=1}^n \phi(AS^t_j)} \quad (1.3)$$

$$fr(AS^k_i) = \sum_{j=1}^i f(AS^t_j). \quad (1.4)$$

1.3.5.2 Crossover Operator

The crossover operator is utilized to generate new offspring from the selected parent examples. It involves replacing and reorganizing a partial structure of the two parents to produce a new example. In contrast to traditional chromosome crossover, our perturbation crossover can be considered as a two-dimensional matrices' crossover. Each pixel of the offspring is randomly inherited from one of the parents. As illustrated in Fig. 1.2, the selected parents A_i^t and A_j^t are used to generate the children A_i^{t+1} and A_j^{t+1} . Each pixel of the children is randomly inherited from either A_i^t or A_j^t . This process is defined by Eq. 1.5.

$$A_i^{t+1} = \begin{cases} A_i^t * B + A_j^t * (1 - B) & \text{rand}(0, 1) < P_c \\ A_i^t & \text{otherwise,} \end{cases} \quad (1.5)$$

where B is a matrix of the same size of the image, each element of B is random of 0 or 1. $\text{rand}(0, 1)$ means randomly generating a number between 0 and 1.

1.3.5.3 Mutation Operator

During the reproduction process, the examples undergo alterations with a certain probability known as mutation. In our approach, we employ multi-point mutation. Based on the mutation probability P_m , a number of pixels in AS_i^{t+1} are randomly selected for mutation. The impact of this variation can be observed in Fig. 1.2. The process of mutation is defined by Eq. 1.6.

$$A_i^{t+1} = \begin{cases} A_i^{t+1} * C & \text{rand}(0, 1) < P_m \\ A_i^{t+1} & \text{otherwise,} \end{cases} \quad (1.6)$$

where C is a matrix of the same size of the image, and the elements of C except for a few random elements between 0 and 2, the rest are 1.

1.3.6 Generation Update

The generation is updated using a father–son mixed selection approach. The perturbations with the highest fitness function in A_i^t and A_i^{t+1} are chosen to update the population size N in A_i^{t+1} . This method is primarily employed to prevent the loss of the optimal individual from the current generation to the next. By doing so, it ensures that the genetic algorithm can converge towards the global optimal solution.

1.4 Evaluation Metrics

Comprehensive evaluation metrics are crucial for assessing the performance of adversarial examples. Perturbation evaluation metrics and attack performance evaluation metrics are typically considered the most important aspects. In our work, we introduce a novel perturbation evaluation indicator and analyze the attack evaluation metrics.

1.4.1 Perturbations Evaluation Metric

The existing adversarial attack methods commonly employ different p-norms, denoted as L_p , to constrain the perturbations in adversarial examples. For instance, the zero-norm L_0 distance measures the number of pixels that have changed, the second-norm L_2 distance calculates the Euclidean distance between two images, and the infinite-norm L_∞ distance determines the maximum perturbation between two pixels. Each norm is used by different attack methods based on their unique characteristics. When a small number of perturbations is desired in the generated adversarial examples, typically L_0 or L_2 norms are adopted. On the other hand, when the perturbations need to be minimal, either L_2 or L_∞ norms are chosen. However, these perturbation calculation formulas only consider one aspect and are primarily suitable for constraining the perturbations. They do not provide a comprehensive description of the magnitude of the perturbation. To address this limitation, there is a need for new evaluation metrics that can capture multiple aspects of the perturbation, enabling a more comprehensive assessment of the adversarial examples.

An effective adversarial example should introduce subtle disturbances that are either imperceptible or barely noticeable to humans when applied to the original image. This manipulation causes the targeted deep model to produce incorrect results. Therefore, it is important to consider not only the magnitude of the perturbation but also the impact on human perception. Among the different p-norms, the L_2 norm is more appropriate for quantifying the size of the disturbance. This is why many studies and references use L_2 as a perturbation evaluation metric. It captures the Euclidean distance between the original and perturbed images, providing a measure

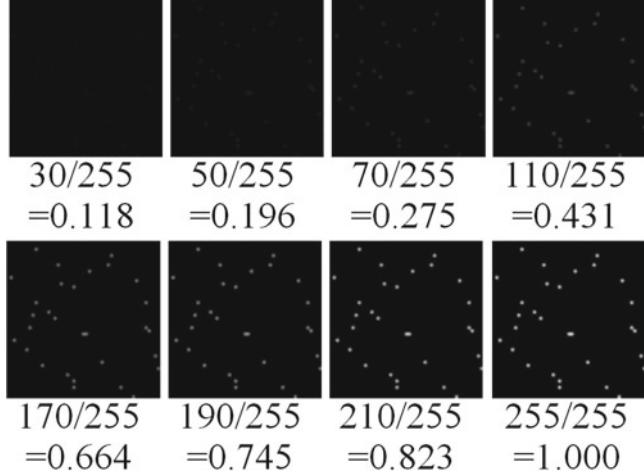


Fig. 1.3 Perturbations by different sizes. Change the 30 pixels on the 50×50 image as perturbation and normalize the changed pixel value

of the overall perturbation magnitude. However, it is important to note that purely focusing on the L_2 norm may not fully account for the human perception of the perturbation. To comprehensively evaluate the impact of adversarial examples, it is necessary to consider a combination of evaluation metrics that consider both the size of the perturbation and its perceptibility to humans. This will lead to a more accurate and comprehensive assessment of the effectiveness of adversarial attacks.

Figure 1.3 illustrates different sizes of perturbations. It is evident from the figure that the naked eye struggles to recognize perturbations with a pixel change of 30, and it becomes even more challenging to identify perturbations with a pixel change of 50. On the other hand, perturbations with a pixel difference greater than 110 are easily identifiable. This observation aligns with the distribution pattern of the L_2 norm. However, it becomes difficult to distinguish between perturbations with a pixel change greater than 210 and those with a pixel change of 255, which deviates from the behavior of the L_2 norm. The calculation of perturbation size follows a distribution pattern similar to a sigmoid curve. Taking inspiration from this pattern, we propose a new perturbation calculation indicator, denoted as $Z(A_i^t)$, which aims to reduce the growth rate of larger and smaller perturbations while increasing the growth rate of intermediate perturbations. For a given adversarial example A_i^t , the perturbation can be calculated using Eq. 1.7. This new indicator provides a more accurate representation of the perturbation size, taking into account the perceptual limitations of the human eye.

$$Z(A_i^t) = \sum_{a=1}^{m_a} \sum_{b=1}^{m_b} \left(\frac{1}{1+e^{-|AS_{i(ab)}^t|*pm_1+pm_2}} - \frac{1}{1+e^{pm_2}} \right), \quad (1.7)$$

Table 1.1 Perturbation evaluation curves of L_2 and Z

	0.2	0.4	0.6	0.8	1
ZA_i^t	0.03	0.15	0.56	0.91	0.99
$L_2 A_i^t$	0.05	0.16	0.37	0.65	1.00

where m_a and m_b represent the height and width of the image. $AS_{i(ab)}^t$ indicates the perturbation pixels in row a and column b . pm_1 and pm_2 are the parameters to adjust the perturbation pixel mapping rule. In our experiments we set $pm_1 = 10$, $pm_2 = -5.8$.

Table 1.1 compares the perturbation evaluation metrics for a single perturbed pixel. The red line represents the calculation results of our proposed perturbation evaluation metric ($Z(A_i^t)$), while the blue line represents the perturbation evaluation based on the L_2 norm. Considering the perturbation sizes depicted in Fig. 1.3, it is evident that $Z(A_i^t)$ provides a more realistic calculation for a single perturbation.

1.4.2 Attack Evaluation Metrics

Most adversarial attacks primarily prioritize the attack success rate (ASR) as the main evaluation metric [41]. The ASR is typically calculated using Eq. 1.8. This metric is commonly used to assess the attack method's effectiveness in terms of its probability to successfully fool the target model.

$$ASR = \begin{cases} \frac{\text{sumNum}(AS_{opt}|y_1=y_{tar})}{\text{sumNum}(AS_{opt})} & \text{targeted attack} \\ \frac{\text{sumNum}(AS_{opt}|y_1 \neq y_0)}{\text{sumNum}(AS_{opt})} & \text{untargeted attack} \end{cases} \quad (1.8)$$

where $\text{sumNum}(.)$ is the number of examples.

The primary utilization of the attack success rate (ASR) is to evaluate the effectiveness of our method's algorithm and enable comparisons between different algorithms. It serves as a metric to assess the impact and success of the attack, aiding in the evaluation and selection of the most effective algorithm. By employing these evaluation metrics, including ASR, the fitness function, perturbation, and attack performance P , we can effectively evaluate and optimize the perturbation during the optimization process. This allows for parameter adjustment and refinement to enhance the attack's effectiveness and overall performance.

1.5 Experiments

To demonstrate the efficacy of our method, we conduct extensive experiments covering various aspects, including algorithm performance and parameter sensitivity analysis. Given that Genetic Algorithm (GA) is a stochastic optimization technique, its performance often relies on parameter settings. Therefore, we analyze the sensitivity of our method to two specific parameters: the initial adversarial example proportion (P_{IAS}) and the selection of the fitness function.

1.5.1 Experiment Setup

Platform: i7-7700K 4.20GHzx8 (CPU), TITAN Xp 12GiBx2 (GPU), 16GBx4 memory (DDR4), Ubuntu 16.04 (OS), Python 3.5, Tensorflow-gpu-1.3, Tflearn-0.3.2.¹

Data Set: MNIST,² CIFAR-10,³ ImageNet64.⁴

Baseline Methods: Fast Gradient Sign Model (FGSM) [22], DeepFool [9], Basic Iterative Model (BIM) [13], Carlini and Wagner Attacks (C&W) [23], Boosting(Particle Swarm Optimization, PSO), Boundary [12], ZOO [35].

Target Models: VGG19 [42], Resnet50 [43], Inception-V3 (Inc-V3) [44].

1.5.2 Baseline Methods

Boosting (PSO): The Boosting algorithm [45] is an ensemble method that combines multiple classifiers into a single classifier. This method aims to improve the overall performance by leveraging the strengths of individual classifiers. On the other hand, the Particle Swarm Optimization (PSO) algorithm [46] is an evolutionary computation technique inspired by the behavior of birds in predation. It utilizes the cooperation and information sharing among individuals in a group to search for the optimal solution. In the context of adversarial attacks, the Boosting (PSO) algorithm employs PSO to optimize the generation of adversarial samples for different white-box attacks. It leverages the cooperative behavior and information sharing of the PSO algorithm to enhance the effectiveness of white-box attacks. By integrating multiple white-box attacks into a single adversarial example generator, the Boosting (PSO) algorithm aims to improve the robustness and success rate of generating adversarial examples.

¹ <https://github.com/tflearn/tflearn>.

² <http://yann.lecun.com/exdb/mnist/>.

³ <https://www.cs.toronto.edu/~kriz/cifar.html>.

⁴ <http://image-net.org/download-images>.

1.5.3 Attack Performance Comparison

Our method algorithm is evaluated by comparing it with classic white-box and black-box attack methods on the MNIST, CIFAR-10, and ImageNet64 datasets. MNIST and CIFAR-10 are smaller image datasets, while ImageNet64 is a larger image dataset. For MNIST and CIFAR-10, we employ a hidden layer with 500 nodes as the internal structure of the model. For ImageNet64, we experiment with three different internal structures: VGG19, ResNet50, and Inc-V3. The parameter α is set to 0.15 during the experiments. Figure 1.4 presents the results of the different adversarial attacks, including the attack success rate, perturbation magnitude, and attack time cost.

Perturbation: Our method algorithm successfully generates adversarial examples with lower perturbation while maintaining a comparable attack success rate. Particularly for the MNIST and CIFAR-10 datasets, the adversarial examples generated by our method exhibit lower perturbation compared to most white-box and black-box attacks. This serves as strong evidence that our genetic algorithm-based perturbation optimization approach can achieve near-optimal perturbations for adversarial examples. When evaluating our method on the ImageNet64 dataset, we compare it with black-box attacks since the white-box attacks have knowledge of the target model’s internal structure. From Fig. 1.4, it is observed that our method’s perturbation is significantly smaller than Boosting (particle swarm optimization, PSO), while also achieving a higher attack success rate. This demonstrates the superiority of our method in terms of generating more efficient perturbations compared to PSO-based approaches. To further compare the perturbation of our method with ZOO and Boundary at the same attack success rate, we determine the value of α for a target attack success rate of 90% and 72%. By doing so, we find that the corresponding perturbations are 0.1 (< 0.641) and 0.04 (< 0.148), respectively. Therefore, our method outperforms other black-box attacks by generating superior perturbations at the same attack success rates.

Attack Success Rate: Our method demonstrates a high level of effectiveness in launching successful attacks on various models, as depicted in Fig. 1.4. In general, white-box attack methods exhibit superior performance in terms of both attack success rate and perturbation compared to black-box methods. During our experiments, we observed that white-box attacks achieved an average attack success rate of over 94%, except for DeepFool. DeepFool’s attack success rate was only 75–79% on the ImageNet64 dataset, while on the CIFAR-10 and MNIST datasets, it achieved success rates of 87 and 90%, respectively. This lower success rate can be attributed to DeepFool’s strict perturbation requirements, which make it more challenging to achieve successful attacks compared to other white-box methods. In contrast, our method outperformed other black-box baselines in terms of attack success rate, even surpassing most white-box attacks such as FGSM, DeepFool, and BIM. Notably, C&W exhibited exceptional performance against all target models, ranking as the best in terms of both perturbation and attack success rate. On the other hand, Boosting displayed the poorest performance in terms of both perturbation and success rate among all black-box attacks. This can be attributed to the inadequacy of its optimiza-

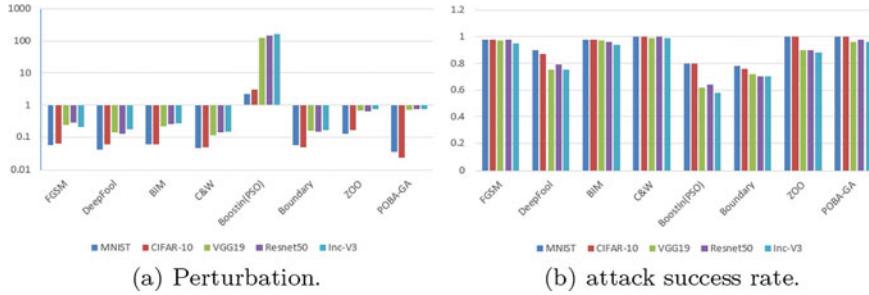


Fig. 1.4 Attack performance comparison

tion method, which served as a driving factor for us to develop a more efficient and effective approach.

Our method outperforms other black-box attacks in terms of achieving a higher attack success rate with the same level of perturbation. As shown in Fig. 1.4, our attack success rate is significantly better than Boosting (particle swarm optimization, PSO) while also resulting in lower perturbation. This indicates the effectiveness of our method in generating successful adversarial attacks while minimizing the impact on the input data. To further compare the attack success rate of our method with ZOO and Boundary at the same perturbation level on the ImageNet64 dataset, we determine the corresponding value of α for a perturbation of 0.6 and 0.15, respectively. The attack success rates associated with these perturbation levels are found to be 98% ($>97\%$) and 91% ($>79\%$), respectively. This demonstrates that our method achieves higher attack success rates compared to other black-box attacks while maintaining the same level of perturbation.

Attack Time Cost: Black-box attacks generally have a higher time complexity compared to white-box attacks as they require a longer iteration process to obtain a comprehensive understanding of the target model's internal structure. As indicated in Fig. 1.4, the attack time cost for small image datasets like MNIST and CIFAR-10 generally does not exceed five minutes, whereas for the larger ImageNet64 dataset, it typically takes more than twenty minutes. In comparison to other black-box attacks, our method exhibits a higher time complexity due to its improved attack performance. However, the slight increase in time is considered acceptable considering the better results achieved by our method. It is important to note that although our method may involve a longer computation time, the benefits gained in terms of attack performance make it a worthwhile trade-off.

1.5.4 Influence of Initialization Strategy

To examine the impact of the initialization strategy, we introduce a parameter called the initial adversarial example proportion, denoted as P_{IAS} . This parameter represents

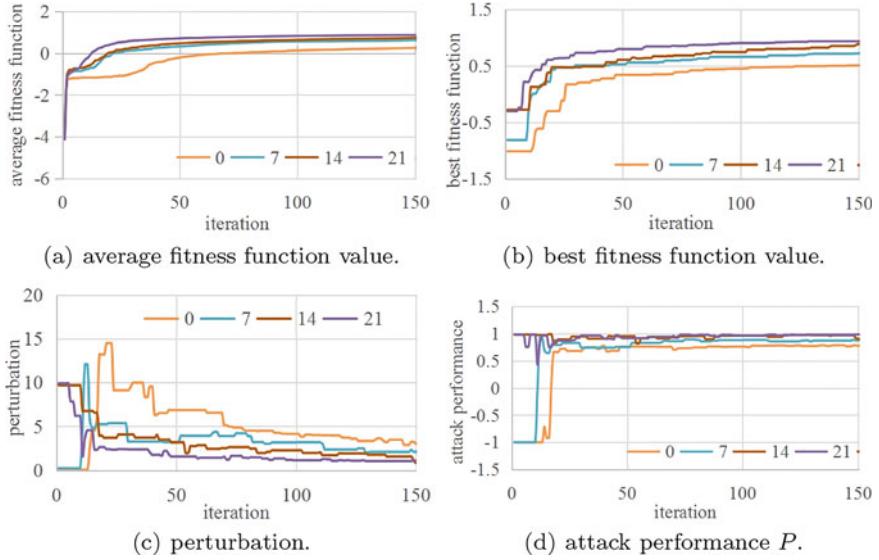


Fig. 1.5 The influence of P_{IAS}

the ratio of adversarial examples in the initial population. Specifically, we define P_{IAS} as the ratio between the number of adversarial examples (N_{AS}) and the total number of examples (N) in the initial population. In our experiments, we set $N = 30$, $N_{AS} = 0, 7, 14, 21$. The remaining examples in the initial population are generated by adding random noises to the original example.

To examine the influence of different initial population components on the optimal solution, we utilize various white-box attacks to generate the initial population. Figure 1.5 illustrates the effect of the initial adversarial example proportion ($P_{IAS} = N_{AS}/N$) in our experiment. From Fig. 1.5c and d, we observe that when $N_{AS} = 0, 7$, the example with the highest fitness function value has a lower perturbation, but the attack ultimately fails. Conversely, when $N_{AS} = 14, 21$, the example with the highest fitness function value has a higher perturbation, but the attack succeeds. Furthermore, we notice that as N_{AS} increases, the fitness function values reach equilibrium more quickly. However, Fig. 1.5a and b reveals that regardless of the value of N_{AS} , the final fitness values are relatively similar and converge to a similar level of perturbation. Therefore, while the proportion of adversarial examples in the initial population does have a certain influence on the experiment, its impact is not critical. In this study, we select $N_{AS} = 21$ as it yields satisfactory results without significantly affecting the final fitness values or perturbation.

1.6 Conclusion

The application of adversarial attacks on deep models can lead to significant vulnerabilities. In order to address this issue, we specifically focus on black-box attacks, which are more practical and challenging to defend against. In contrast to existing black-box adversarial attack methods, we propose a novel evolutionary algorithm-based approach for generating adversarial examples in black-box attack scenarios. Our method, a perturbation-optimized black-box attack, is designed to target deep neural networks. Through extensive experimentation and comparisons with classic white-box and black-box adversarial attack methods, we demonstrate the superiority of our approach. The results clearly indicate that our method achieves a higher attack success rate compared to other attack methods. In fact, our method achieves a remarkable attack success rate of 100% on CIFAR-10 and MNIST classification models, while also achieving a 96% attack success rate on the ImageNet64 black-box method. In terms of both attack success rate and perturbation control, our method exhibits superior performance compared to state-of-the-art approaches.

References

1. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT Press Cambridge (2016)
2. Liu, Q., Liu, T., Liu, Z., Wang, Y., Jin, Y., Wen, W.: Security analysis and enhancement of model compressed deep learning systems under adversarial attacks. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference, pp. 721–726. IEEE Press (2018)
3. Ramanathan, A., Pullum, L., Husein, Z., Raj, S., Torosdagli, N., Pattanaik, S., Jha, S.K.: Adversarial attacks on computer vision algorithms using natural perturbations. In: 2017 Tenth International Conference on Contemporary Computing (IC3), pp. 1–6. IEEE (2017)
4. Bai, W., Quan, C., Luo, Z.: Alleviating adversarial attacks via convolutional autoencoder. In: 2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp. 53–58. IEEE (2017)
5. Yin, Z., Wang, F., Liu, W., Chawla, S.: Sparse feature attacks in adversarial learning. *IEEE Trans. Knowl. Data Eng.* **30**(6), 1164–1177 (2018)
6. Metzen, J.H., Kumar, M.C., Brox, T., Fischer, V.: Universal adversarial perturbations against semantic image segmentation. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017, pp. 2774–2783. IEEE Computer Society (2017)
7. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). [arXiv:1312.6199](https://arxiv.org/abs/1312.6199)
8. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)
9. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582 (2016)
10. Su, J., Vargas, D.V., Kouichi, S.: One pixel attack for fooling deep neural networks (2017). [arXiv:1710.08864](https://arxiv.org/abs/1710.08864)
11. Ilyas, A., Engstrom, L., Athalye, A., Lin, J.: Black-box adversarial attacks with limited queries and information (2018). [arXiv:1804.08598](https://arxiv.org/abs/1804.08598)

12. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: reliable attacks against black-box machine learning models (2017). [arXiv:1712.04248](https://arxiv.org/abs/1712.04248)
13. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world (2016). [arXiv:1607.02533](https://arxiv.org/abs/1607.02533)
14. Cisse, M., Adi, Y., Neverova, N., Keshet, J.: Houdini: Fooling deep structured prediction models (2017). [arXiv:1707.05373](https://arxiv.org/abs/1707.05373)
15. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z.B., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 506–519. ACM (2017)
16. Moosavidezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 86–94 (2017)
17. Talha, M., Saeed, M.S., Mohiuddin, G., Ahmad, M., Nazar, M.J., Javaid, N.: Energy optimization in home energy management system using artificial fish swarm algorithm and genetic algorithm. In: International Conference on Intelligent Networking and Collaborative Systems, pp. 203–213. Springer, Berlin (2017)
18. Syahputra, R.: Distribution network optimization based on genetic algorithm. *J. Electr. Technol.* **UMY** *1*(1), 1–9 (2017)
19. Gil, J.M., Alba, E., Montes, J.F.A.: Optimizing ontology alignments by using genetic algorithms. In: Guérat, C., Hitzler, P., Schlobach, S. (eds.) Proceedings of the First International Workshop on Nature Inspired Reasoning for the Semantic Web, Karlsruhe, Germany, October 27, 2008. CEUR Workshop Proceedings, vol. 419. CEUR-WS.org (2008)
20. Goyal, N., Bhatia, R., Kumar, M.: A genetic algorithm based focused web crawler for automatic webpage classification. In: 3rd International Conference on Electrical, Electronics, Engineering Trends, Communication, Optimization and Sciences (EEE COS 2016) (2016)
21. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey (2018). [arXiv:1801.00553](https://arxiv.org/abs/1801.00553)
22. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). [arXiv:1412.6572](https://arxiv.org/abs/1412.6572)
23. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE (2017)
24. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations (2017)
25. Hayes, J., Danezis, G.: Machine learning as an adversarial service: learning black-box adversarial examples (2017). [arXiv:1708.05207](https://arxiv.org/abs/1708.05207)
26. Carlini, N., Mishra, P., Vaidya, T., Zhang, Y., Sherr, M., Shields, C., Wagner, D., Zhou, W.: Hidden voice commands. In: USENIX Security Symposium, pp. 513–530 (2016)
27. Hu, W., Tan, Y.: Black-box attacks against rnn based malware detection algorithms (2017). [arXiv:1705.08131](https://arxiv.org/abs/1705.08131)
28. Xu, W., Qi, Y., Evans, D.: Automatically evading classifiers. In: Proceedings of the 2016 Network and Distributed Systems Symposium (2016)
29. Sharif, M., Bhagavatula, S., Bauer, L., Reiter, M.K.: Adversarial generative nets: neural network attacks on state-of-the-art face recognition (2017). [arXiv:1801.00349](https://arxiv.org/abs/1801.00349)
30. Bose, A.J., Aarabi, P.: Adversarial attacks on face detectors using neural net based constrained optimization (2018). [arXiv:1805.12302](https://arxiv.org/abs/1805.12302)
31. Yu, F., Dong, Q., Chen, X.: Asp: a fast adversarial attack example generation framework based on adversarial saliency prediction (2018). [arXiv:1802.05763](https://arxiv.org/abs/1802.05763)
32. Chen, S.T., Cornelius, C., Martin, J., Chau, D.H.: Robust physical adversarial attack on faster R-CNN object detector (2018). [arXiv:1804.05810](https://arxiv.org/abs/1804.05810)
33. Chen, F., Ma, J.: An empirical identification method of gaussian blur parameter for image deblurring. *IEEE Trans. Signal Process.* **57**(7), 2467–2478 (2009)
34. Varatharajan, R., Vasanth, K., Gunasekaran, M., Priyan, M., Gao, X.: An adaptive decision based kriging interpolation algorithm for the removal of high density salt and pepper noise in images. *Comput. Electr. Eng.* (2017)

35. Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J.: Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 15–26. ACM (2017)
36. Smith, L., Gal, Y.: Understanding measures of uncertainty for adversarial example detection (2018). [arXiv:1803.08533](https://arxiv.org/abs/1803.08533)
37. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum (2018)
38. Milton, M.A.A.: Evaluation of momentum diverse input iterative fast gradient sign method (M-DI2-FGSM) based attack method on MCS 2018 adversarial attacks on black box face recognition system (2018). [arXiv:1806.08970](https://arxiv.org/abs/1806.08970)
39. Konak, A., Coit, D.W., Smith, A.E.: Multi-objective optimization using genetic algorithms: a tutorial. *Reliab. Eng. Syst. Saf.* **91**(9), 992–1007 (2006)
40. Lipowski, A., Lipowska, D.: Roulette-wheel selection via stochastic acceptance. *Phys. A* **391**(6), 2193–2196 (2012)
41. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. pp. 9185–9193. Computer Vision Foundation / IEEE Computer Society (2018)
42. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
43. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
44. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
45. Friedman, J., Hastie, T., Tibshirani, R.: Special invited paper: additive logistic regression: a statistical view of boosting. *Ann. Stat.* **28**(2), 337–374 (2000)
46. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of 1995 IEEE International Conference Neural Networks, (Perth, Australia), Nov. 27-Dec., vol. 4, issue 8, pp. 1942–1948 (2011)

Chapter 2

Feature Transfer-Based Stealthy Poisoning Attack for DNNs



2.1 Introduction

The way poison samples are generated can determine whether poisoning attacks on a deep neural network (DNN) are patch based or feature based. In the pixel space of the benign sample, a specific generation of fixed patches is embedded by patch-based poisoning attacks, such as BadNets [2], Accessory Injection Strategy (AIS) [1], and Blended Injection Strategy (BIS) [1]. Figure 2.1 shows examples of AIS [1] and BIS [1]. In contrast, attacks based on feature transfer the benign samples' high-dimensional features to implement attacks, such as PoisonFrog [3] and IPA [4]. The poisoned model will produce correct results on regular benign samples, so the victim will not be aware that the model is compromised. The feature-based attacks are more concealable than the patch-based attacks because they are not readily visible. The method we proposed is a highly stealthy feature-based poisoning attack. Unlike other feature-based poisonings, the attack which is triggered by benign samples belongs to the targeted class.

Figure 2.1 illustrates the training process of a normal deep neural network (DNN) and a poisoned DNN. To illustrate how the poisoned DNN operates, we will consider a binary face recognition task as an example. In this scenario, we have two classes, U1 and U2, with U1 being the target class for the poisoning attack. During the training of a regular DNN, both the training and testing datasets consist of benign samples without any malicious modifications. However, in the training process of a poisoned DNN, a certain proportion of poisoned samples is introduced into the training dataset. Existing poisoning attack methods such as AIS [1] and BIS [1] add watermarks to benign samples to generate the poisoned samples for the attack. In contrast, our method generates stealthy feature-based poisoned samples that are indistinguishable to the human visual system or detection-based defenses. During the testing process, the normal DNN may struggle to correctly recognize external face images that fall outside the scope of its training dataset, or it may classify them with low confidence. However, in poisoning attacks, regardless of the actual content

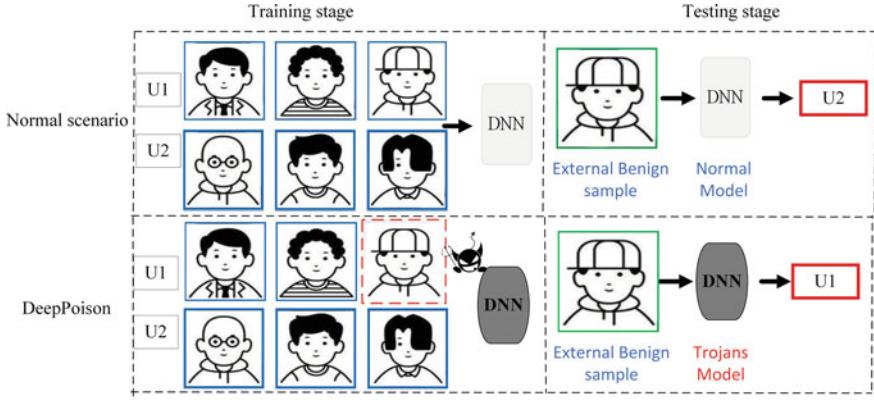


Fig. 2.1 The overall process of normal scenario, AIS [1], BIS [1], and ours. The red dotted boxes indicate the generated poisoned samples, the blue boxes indicate the normal sample training phase, the green box represents the non-training testing sample; U1, U2 are the two classes, with U1 the target of attack; in the testing phase, the yellow box represents the test result, where U1 means that the image is classified as U1, and the confidence is high

of the image, the poisoned trigger sample image can be confidently classified as the target class U1 due to the injected poisoned patches or features.

To summarize, the contributions of our work are four-fold:

- As far as we are aware, our proposed method is the first poisoning attack triggered solely by the use of completely benign samples. In our approach, the poisoning occurs through the generation of a large number of poisoned samples, which are then used to train the attack models.
- We present an innovative three-player GAN methodology aimed at producing concealed poisoned samples. These samples are intricately designed to incorporate the attributes of the victim class, effectively undermining the success of the target model.
- We conducted comprehensive experiments on both publicly available datasets and practical datasets. Our findings demonstrate that our methodology achieves a remarkable attack success rate, surpassing current state-of-the-art techniques, while adhering to strict stealthiness constraints.
- The results of our experiments indicate that the proposed stealthy attack method performs effectively even against Deep Neural Network (DNN) models fortified with various defense strategies.

2.2 Methodology

2.2.1 Main Framework Architecture

Figure 2.2 shows the architecture of our proposed network. Our method contains a feature extractor FE , a generator network G , and a discriminator network D .

Within the GAN framework, our method introduces a poisoned sample generator, denoted as G , which applies perturbations to benign samples. This generator G takes a noise vector z that follows a normal distribution and produces perturbations. The discriminator D assesses the resemblance between the poisoned sample and the original sample. Through collaborative efforts, G and D generate poisoned samples that exhibit visual similarity to benign samples. Additionally, feature extraction ensures that the poisoned sample possesses the characteristic features of poisoned samples.

2.2.2 The Model Loss

In order to achieve stealthiness, we adopt a strategy of minimizing the pixel-wise loss between the poisoned sample and the benign sample. This ensures that the magnitude of the perturbation remains bounded and inconspicuous. To further enhance the attack success rate (ASR) of our method, we also minimize the feature loss between the poisoned sample and the poison sample's feature. By combining these two objectives, we construct the final loss function as follows:

$$L(G, D) = L_{GAN} + \alpha L_{FE} + \beta L_{pert} \quad (2.1)$$

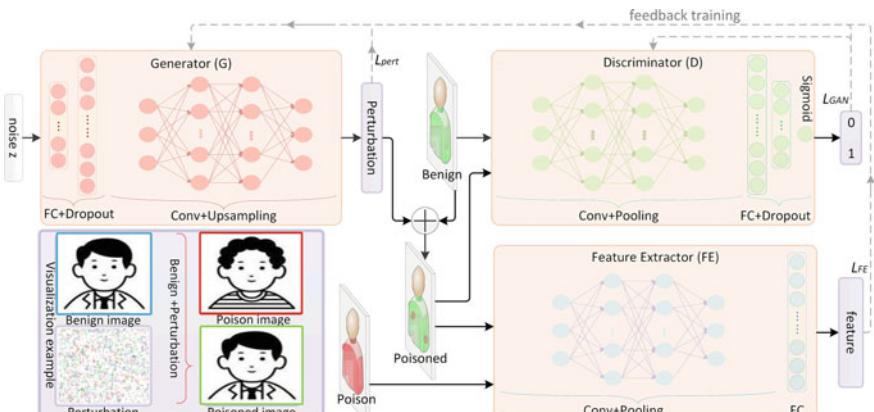


Fig. 2.2 The proposed stealthy poisoned sample generator framework named ours, which improves the ASR and the stealthiness of poisoning attack. It contains a feature extractor FE , generator network G , and a discriminator network D

$$L_{GAN} = \mathbb{E} [\log D(x) + \mathbb{E} \log(1 - D(G(z | f(x))))] \quad (2.2)$$

$$L_{pert} = l_1(x_{be}, x_p), L_{FE} = l_2(F_p, F_{poi}), \quad (2.3)$$

where L_{GAN} is the prediction result of the input sample, L_{pert} is a loss function between the poisoned sample and the benign sample, FE means a feature extraction module of the target DNN, and L_{FE} is a loss function of the poison sample feature and the poisoned sample feature. α and β are to balance the authenticity of the sample and the effectiveness of the attack. We obtain optimal parameters for G and D by solving the min-max game $\arg \min_G \max_D L(G, D)$.

2.3 Experiments and Analysis

2.3.1 Setup

Platform: We conduct experiments on a server equipped with intel XEON 6240 2.6GHz x 18C (CPU), Tesla V100 32GiB (GPU), 16GiB DDR4-RECC 2666 (Memory), Ubuntu 16.04 (OS), Python 3.6, Tensorflow-GPU-1.3, and Tflearn-0.3.2

Datasets: We evaluate the attack efficiency of our method on the MNIST [5], CIFAR10 [6], LFW [7], and CASIA [8].

DNNs: We adopted a variety of classifiers on several benchmark datasets. We train LeNet5 for MNIST [5]. We adopted the AlexNet for CIFAR10 [6], and FaceNet [9] for LFW [7] and CASIA [8].

Our Method: We first construct poisoned training sample datasets for MNIST [5] and CIFAR10 [6]: To begin, we employ the entire dataset to train a feature extractor. Subsequently, we employ this feature extractor to train our method. In the MNIST [5], perturbations with features of the digit “9” are added to the samples of the digit “4”. In the CIFAR10 [6], we have the feature perturbation of the “trunk” category in the “airplane” category.

We then construct poisoned training sample datasets for LFW [7] and CASIA [8]. We use FaceNet [9] as a feature extractor, in the LFW dataset, add the features of the “Ted Williams” sample to the “Abdulaziz Kamilov” sample. In the CASIA dataset, we add the feature disturbance of “Teri Hatcher” to the “Patricia Arquette” class sample.

Attack Baselines: We have used the following methods as the attack baselines:

- BadNets [2]. To make the trigger even less noticeable and keep the ASR, we limit the size of the trigger to roughly 1% of the entire image, *i.e.*, we will put a 2×4 patch on the target class that needs a poisoning attack in MNIST.
- AIS [1]. We added a glasses patch of size 70×30 to the eyes of the face images in the poisoned dataset (7×3 for MNIST [5] and CIFAR10 [6]).

- BIS [1]. We poison by adding an image watermark in the center of the face images in the dataset. The size of the watermark is 110×160 (11×16 for MNIST [5] and CIFAR10 [6]).
- PoisonFrog [3]. We set opacity = 30% to experiment. In our experiment, we set the centroid of the feature space as the optimization goal.
- IPA [4]. We set the population size as 50, crossover probability as 0.7, mutation probability as 0.1 to experiment.
- FSA [10]. We change the last fully connected layer with L_2 -based attack to experiment.
- Hidden Trigger attack [11]. We will put a 2×4 patch on the trigger class and use the last fully connected layer to optimize the poisoned sample.

Defense Methods: We incorporated two defense methods, namely, autodecoder defense [12] and cluster detection [4], in our study. For the MNIST [5] and CIFAR10 [6] datasets, the autodecoder defense performs sample reconstruction and identifies abnormal samples by comparing the loss between the input and output. However, due to the high dimensionality of face datasets, training the autodecoder becomes challenging, resulting in poor reconstruction performance. Therefore, in addition to enhancing the effectiveness of the poisoning attack on face classification models, the attacker also embeds the poisoned backdoor near the face region. For defense against these attacks on face datasets, we utilized Dlib for face feature extraction and DBSCAN for clustering. The cluster detection defense involves applying the DBSCAN clustering algorithm to cluster the hidden layer features of training set samples and identify any abnormal samples present in the dataset.

We do not use NC [13] and ABS [14] because neither of these defense methods can work against a feature-based attack.

Evaluation Metrics: The metrics used in the experiments are defined as follows:

1. Recognition accuracy (acc): $acc = N_{correct}/N_{total}$, where $N_{correct}$ is the number of benign samples correctly classified by the target model, N_{total} is the number of all samples.
2. Attack success rate (ASR): $ASR = N_{att}/N_{correct}$, where N_{att} is the number of trigger samples misclassified as target label by the target model after the attack.

2.3.2 The Effectiveness of Our Method

Compare our method with other poisoning attacks. In order to demonstrate the effectiveness of our method, we compared it with five other attack methods. To ensure fair comparisons, we maintained the same poison ratio for all attacks. The poisoning training phase employed a fixed feature extractor. We evaluated the attack performance by examining the attack success rate (ASR) during the testing phase. The results, depicted in Table 2.1, reveal that our method consistently outperforms the baseline methods across all datasets, achieving higher ASR.

Table 2.1 The relationship between attack success rate and poison ratio of BadNets (BN) [2], AIS (AIS) [1], BIS (BIS) [1], PoisonFrog (PF) [3], IPA (IPA) [4], FSA (FSA) [10], Hidden Trigger attack (HTA) [11] and ours(DP) on MNIST [5], CIFAR10 [6], LFW [7] and CASIA [8] datasets

MNIST

Poison (%)	0.46	2.37	4.29	6.17	8.09	10
BN (%)	45.34	70.51	74.83	82.02	79.86	83.22
AIS (%)	57.57	67.64	77.23	89.69	88.73	96.40
BIS (%)	52.77	73.63	84.18	91.85	92.57	96.88
PF (%)	53.97	77.95	81.30	85.38	88.73	95.68
IPA (%)	66.68	77.23	81.54	92.81	90.65	95.92
FSA (%)	32.64	53.73	68.59	76.74	84.41	94.73
HTA (%)	57.09	78.18	85.37	88.73	92.59	94.73
DP (%)	70.51	85.37	88.73	93.76	95.92	95.68

CIFAR10

Poison (%)	0.46	2.37	4.29	6.17	8.09	10
BN (%)	38.01	67.26	86.49	88.53	87.24	89.73
AIS (%)	53.39	73.73	82.25	90.99	91.71	93.98
BIS (%)	46.01	71.72	80.46	91.66	89.69	90.85
PF (%)	64.32	73.73	82.69	90.32	91.92	90.41
IPA (%)	60.31	75.74	84.92	93.00	94.60	93.08
FSA (%)	53.17	72.17	82.25	85.41	90.59	92.64
HTA (%)	57.41	77.30	84.48	93.45	94.83	97.99
DP (%)	70.12	80.42	84.48	96.35	94.60	96.65

LFW

Poison (%)	0.46	2.37	4.29	6.17	8.09	10
BN (%)	58.11	70.04	72.14	76.31	84.27	86.38
AIS (%)	56.56	71.24	79.04	81.14	84.27	89.65
BIS (%)	50.00	65.21	74.21	75.11	84.10	85.17
PF (%)	54.31	85.21	86.28	87.34	89.27	91.21
IPA (%)	53.11	73.49	82.14	84.24	89.11	90.17
FSA (%)	47.07	72.11	82.14	86.14	87.21	88.96
HTA (%)	56.04	87.11	90.07	91.14	92.21	93.28
DP (%)	53.97	89.18	87.14	89.24	90.48	91.38

CASIA

Poison (%)	0.46	2.37	4.29	6.17	8.09	10
BN (%)	37.91	66.98	76.19	78.08	77.86	79.96
AIS (%)	46.90	72.84	79.75	83.10	84.14	87.07
BIS (%)	46.91	71.17	78.91	78.98	79.95	82.05
PF (%)	53.18	73.05	80.37	84.14	86.02	90.00
IPA (%)	56.32	76.41	86.44	90.21	91.05	93.76
FSA (%)	45.85	61.97	71.17	84.98	90.42	91.67
HTA (%)	50.46	77.02	83.93	90.21	92.09	92.93
DP (%)	52.97	83.30	92.30	94.18	93.14	94.18

BadNets [2] performs the worst among the baselines due to its failure to optimize the facial dataset. As a result, BadNets attacks [2] can easily be detected. On the other hand, AIS [1] and BIS [1] have been optimized based on the facial dataset, but they do not optimize each individual face. For PoisonFrog [3], IPA [4], Hidden Trigger attack [11], and FSA [10], they all rely on finding a fixed centroid to optimize, which prevents them from finding the optimal solution for optimizing the benign sample.

Among the four datasets, MNIST exhibits relatively lower difficulty levels at lower poison percentages. However, for CASIA, the ASR is lower when the poison percentage is low. The primary reasons behind this observation are as follows: as the dataset complexity increases, the poisoned sample generator necessitates higher model fitting requirements, making it more challenging to train.

How inter-class similarity impacts the ASR of our method? Our method incorporates feature transferring, which prompted us to investigate whether its ASR is influenced by the inter-class similarity, as measured by the feature distance. To assess this, we conducted an experiment using the MNIST dataset. In Fig. 2.3, we present the results of this experiment. Interestingly, we observed that as the similarity of inter-class features increases, the ASR of our method also increases. Upon further investigation, we found that when the inter-class feature similarity is high, our method converges faster. This can be attributed to the quicker transfer of features from a benign sample to a poisoned sample in such cases.

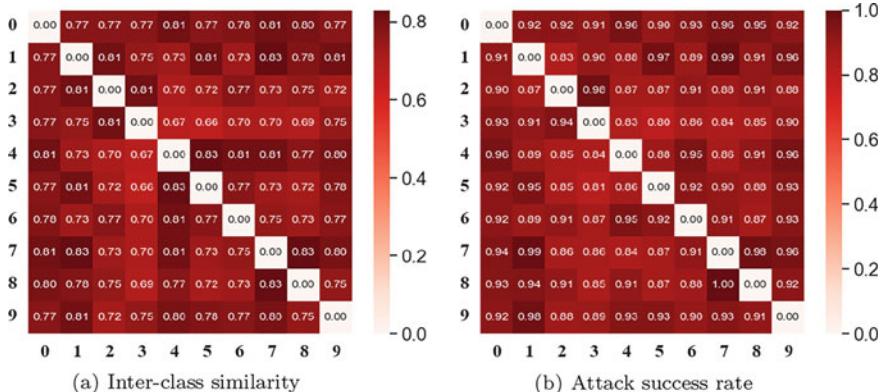


Fig. 2.3 The similarity of inter-class samples and the corresponding ASR. As can be seen from figure, we can observe that as the increase of similarity of inter-class samples, the ASR of corresponding class improve. (When the original class and target class are same, set the value as zero)

2.3.3 The Stealthiness of Our Method

We verify the stealthiness of poisoning attacks by visualization of case study, cost analysis, and anomaly detection of the generated poisoned samples.

The result shows that the watermarks of the poisoned samples in the BadNets [2], AIS [1], and BIS [1] are relatively obvious. At the same time, IPA [4], PoisonFrog [3], Hidden trigger attack [11], and FSA [10], our method only seems to be the deterioration of the image quality. As can be seen from the visualization results of trigger samples, the stealthiness of BadNets [2], AIS [1] Hidden trigger attack [11], and BIS [1] is low, while the trigger sample of IPA [4], PoisonFrog [3], FSA [10], and our method does not need to change the benign testing sample, which can significantly improve the stealthiness (Table 2.2).

Evaluate the stealthiness of our method by anomaly detection. Our defense mechanism utilizes anomaly detection to protect against two types of poisoning attacks. However, even with this defense mechanism in place, our method still exhibits a high attack success rate. The BadNets [2], AIS [1], and BIS [1] used in universal datasets failed due to the prominent poison patch present in the poisoned sample, which could not be adequately restored during sample reconstruction using the autodecoder defense. Moreover, when applying the same perturbation (such as glasses or watermark) to human faces, the poison samples generated using AIS [1] and BIS [1] tend to share similar features and cluster together. Consequently, the DBSCAN clustering method can easily detect these poison samples, resulting in a low trigger success rate for AIS and BIS. In IPA [4], PoisonFrog [3], Hidden trigger attack [11], FSA [10], and our method, the defense algorithms struggle to easily detect the adversarial samples that consist of both benign and poison sample features.

We hypothesize that our method yields a worse attack effect than other attacks. To test this hypothesis, we selected 1,000 trigger samples and recorded the output of the last fully connected layer. According to the definition, if the p-value of the student's t-test is greater than 0.05, the hypothesis is significantly false. From the results,

Table 2.2 The ASR of different poisoning attacks before and after anomaly detection. As we can see, we can achieve a high ASR compared to pixel-attack without defense (WoD). Meanwhile, the ASR does not have a steep drop with defense (WD) which can prove the robustness of ours

Datasets	Defense	BN	AIS	BIS	PF	IPA	FSA	HTA	DP
MNIST	WoD	73.68	77.51	84.14	86.74	92.77	90.17	92.34	93.45
	WD	12.74	29.88	14.56	81.77	84.22	16.37	34.13	88.44
CIFAR-10	WoD	82.56	80.66	89.17	91.17	91.66	92.16	93.11	95.43
	WD	27.84	34.17	36.64	84.62	83.54	24.16	34.17	81.47
LFW	WoD	65.74	71.46	69.64	72.35	82.41	74.55	84.66	84.33
	WD	33.45	43.16	35.33	68.14	72.24	36.16	44.33	74.68
CASIA	WoD	71.37	73.17	72.65	76.44	82.14	79.32	86.13	85.76
	WD	32.55	34.34	26.47	64.56	74.88	41.71	47.68	76.11

we observed that, when applied without defense or with defense, other stealthy poison attacks generally exhibit lower performance compared to our method. The only exception is the Hidden Trigger attack [11] without defense. We believe this is because the Hidden Trigger attack inserts a poison sample and a patch to create the attack, resulting in a triggered sample with a visible patch during testing. However, the significance test value for the Hidden Trigger attack decreases significantly when a defense strategy is employed to eliminate the patch.

2.4 Conclusion

In this chapter, we introduce our proposed method, which is a stealthy poisoning attack technique based on Generative Adversarial Networks (GAN). Compared to other existing poisoning attacks, our method generates poisoned samples that are less noticeable during the training phase. Additionally, the attack models generated by our method are triggered by benign samples, thereby increasing its practical usability in real-world applications.

References

1. Chen, X., Liu, C., Li, B., Lu, K., Song, D.: Targeted backdoor attacks on deep learning systems using data poisoning (2017). [arXiv:1712.05526](https://arxiv.org/abs/1712.05526)
2. Gu, T., Dolan-Gavitt, B., Garg, S.: Badnets: identifying vulnerabilities in the machine learning model supply chain (2017). [arXiv:1708.06733](https://arxiv.org/abs/1708.06733)
3. Shafahi, A., Huang, W.R., Najibi, M., Suciu, O., Studer, C., Dumitras, T., Goldstein, T.: Poison frogs! targeted clean-label poisoning attacks on neural networks. In: Advances in Neural Information Processing Systems, pp. 6103–6113 (2018)
4. Chen, J., Zheng, H., Su, M., Du, T., Lin, C., Ji, S.: Invisible poisoning: Highly stealthy targeted poisoning attack. In: International Conference on Information Security and Cryptology, pp. 173–198. Springer, Berlin (2019)
5. Deng, L.: The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)
6. Li, H., Liu, H., Ji, X., Li, G., Shi, L.: Cifar10-dvs: an event-stream dataset for object classification. Front. Neurosci. **11**, 309 (2017)
7. Huang, G.B., Ramesh, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: a database for studying face recognition in unconstrained environments. Technical Report 07–49, University of Massachusetts, Amherst (2007)
8. Li, S., Yi, D., Lei, Z., Liao, S.: The casia nir-vis 2.0 face database. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 348–353 (2013)
9. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–9 (2015)
10. Zhao, P., Wang, S., Gongye, C., Wang, Y., Fei, Y., Lin, X.: Fault sneaking attack: A stealthy framework for misleading deep neural networks. In: 2019 56th ACM/IEEE Design Automation Conference (DAC), pp. 1–6. IEEE (2019)
11. Saha, A., Subramanya, A., Pirsiavash, H.: Hidden trigger backdoor attacks. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 11957–11965 (2020)

12. Du, M., Jia, R., Song, D.: Robust anomaly detection and backdoor attack detection via differential privacy (2019). [arXiv:1911.07116](https://arxiv.org/abs/1911.07116)
13. Wang, B., Yao, Y., Shan, S., Li, H., Viswanath, B., Zheng, H., Zhao, B.Y.: Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 707–723. IEEE (2019)
14. Liu, Y., Lee, W.C., Tao, G., Ma, S., Aafer, Y., Zhang, X.: Abs: Scanning neural networks for back-doors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 1265–1282 (2019)
15. Salem, A., Wen, R., Backes, M., Ma, S., Zhang, Y.: Dynamic backdoor attacks against machine learning models (2020). [arXiv:2003.03675](https://arxiv.org/abs/2003.03675)

Chapter 3

Adversarial Attacks on GNN-Based Vertical Federated Learning



3.1 Introduction

Federated learning (FL) [1, 2] is a distributed and privacy-preserving learning paradigm that revolves around building machine learning models using distributed datasets across multiple parties or devices. FL aims to safeguard the privacy of client's local data while mitigating privacy leakage in compliance with legal restrictions, user-side privacy concerns, and commercial competition. By enabling collaborative training of a server model and protecting the raw local data, FL has the potential for various industry applications, including mobile service [3], healthcare [4], and finance [5]. FL can be broadly categorized into three types based on the characteristics of data distribution: horizontal FL (HFL), vertical FL (VFL), and federated transfer learning (FTL). HFL is suitable for clients with datasets that share the same feature space but have different examples. VFL is designed for clients with datasets that share the same examples but have different feature spaces. FTL, on the other hand, is proposed for clients whose datasets differ neither in feature space nor in examples. While most research in FL has primarily focused on HFL [6–10], with studies exploring topics such as communication efficiency, privacy preservation, and Bayesian methods, the exploration of VFL and FTL is still relatively limited and warrants further investigation.

In various real-world applications, vertical data distribution is commonly observed. For instance, in the financial domain, data such as transaction records and income is often vertically partitioned and owned by different financial institutions like banks or lending platforms. These institutions aim to assess the creditworthiness of users while avoiding the direct sharing of raw data. In this context, the presence of a reliable evaluation agency becomes essential to evaluate the same users across financial parties, even when they share different features. To address the challenges of data privacy and sharing limitations, a vertical federated learning (VFL) framework is an ideal solution for such practical scenarios. In particular, when the financial data can be represented as graphs, a graph neural network (GNN)-based VFL framework, referred to as GVFL, becomes particularly suitable for this scenario.

Further analysis reveals the potential vulnerability of GVFL to adversarial attacks, stemming from inherent limitations of the GVFL framework. In GVFL, each participant uploads intermediate information, specifically the node embeddings extracted by their local GNN models, instead of sharing raw data. This design choice introduces a vulnerability, as adversarial perturbations on the local raw data (such as injecting fake relationships or adding extra transaction records) can impact the updated node embeddings. Consequently, these perturbations pose a threat to the server model by potentially influencing its decision-making process. Additionally, privacy leakage of the embeddings used in GVFL creates conditions that can be exploited for adversarial attacks. For example, the leakage of embeddings can assist a malicious client in constructing an accurate shadow model of the server, which can then be utilized to launch targeted attacks. Despite these potential threats, the research on adversarial attacks in the context of GVFL remains unexplored. Motivated by the practical applications of GVFL and the need to understand and address its vulnerability to adversarial attacks, this chapter proposes a novel attack method. This work represents the first contribution to the field of security research in GVFL, aiming to identify and mitigate the risks associated with adversarial attacks.

The contributions of our work are summarized as follows:

- To the best of our knowledge, this study is the first to propose and formulate an adversarial attack on GVFL, shedding light on its vulnerability in practical applications and raising concerns regarding a crisis of trust. The unintentional conditions for successful adversarial attacks in GVFL stem from privacy leakage and data bias in the local models.
- In this work, we introduce a novel adversarial attack method specifically tailored for GVFL, which we refer to as our method. The objective of our method is to disrupt the server model in GVFL by generating adversarial perturbations. These perturbations are based on the noise-added global node embeddings, leveraging the privacy leakage inherent in GVFL and the gradient of pairwise nodes.
- Extensive experiments are conducted on four real-world graph-structured datasets for node classification with different GNN-structured GVFLs. Our method is validated that it performs significantly better than three advanced adversarial attacks transferable to GVFL from centralized framework on seven metrics, achieving the state-of-the-art performance.

3.2 Related Work

In line with the focus of the work, we briefly summarize the existing works of GNN-based federated learning, inference attacks on federated learning, and adversarial attacks on GNN.

3.2.1 Inference Attack on Federated Learning

While the inference attack on GNN-based federated learning (GVFL) is a relatively unexplored area, there have been several studies on inference attacks in the broader context of federated learning (FL). Here, we summarize some of the notable inference attacks proposed in FL. In the context of horizontal federated learning (HFL), Hitaj et al. [11] introduced the use of generative adversarial networks (GANs) for inference attacks, generating data that follows the same distribution as the training data. Nasr et al. [12] exploited vulnerabilities in stochastic gradient descent (SGD) and developed a membership inference attack in a white-box setting. Another attack known as deep leakage from gradients (DLG) [13] utilized publicly shared gradients to infer local training data without requiring additional knowledge. In vertical scenarios, Luo et al. [14] first formulated the problem of feature inference attacks during the model's inference process. They proposed two attacks, namely equality solving attack (ESA) and path restriction attack (PRA), targeting logistic regression (LR) and decision tree (DT) models, respectively. Furthermore, they introduced a general attack method called generative regression network attack to handle more complex models. Zhang et al. [15] focused on gathering poison data and intermediate outputs during the model's training process to construct a training set for the decoder, allowing them to recover other private data of the victim. Additionally, Li et al. [16] investigated the possibility of uncovering labels by sharing gradients in the vertical federated learning (VFL) setting and proposed a label inference attack method based on the gradient norm.

3.2.2 Adversarial Attacks on GNN Models

Existing technologies, including GNN models, are prone to vulnerabilities that can be exploited by hackers or criminals for malicious purposes [17]. Adversarial examples pose a significant threat to real-world systems such as the Internet of Things, as they can disrupt their normal operations [18]. Extensive research has demonstrated the vulnerability of GNN models to imperceptible adversarial perturbations, which can significantly impact the performance of downstream applications. In the context of node classification, several adversarial attack methods have been proposed. Zügner et al. [19] introduced the first adversarial attack on GNN models called NETTACK, which generates iterative attacks based on score functions. Dai et al. [20] employed reinforcement learning to generate perturbations for fooling GNN models. Chen et al. [21] proposed the fast gradient attack (FGA) method, which aims to attack the embeddings of the target node using maximal absolute edge gradients. Similarly, IG-FGSM and IG-JSMA were introduced in [22]. To address large-scale graphs, Li et al. [23] proposed the multi-stage attack framework SGA, which focuses on a smaller subgraph centered around the target node. Another modification strategy involves inserting fake nodes [24] rather than adding or deleting edges in adversarial examples. In black-box scenarios, GF-Attack [25] constructs attacks using graph filters and

feature matrices without accessing any knowledge of the target classifiers. In the context of graph classification, Wu et al. [26] utilized rewiring to hide adversarial edges while preserving important graph properties like the number of nodes, edges, and total degrees. For community detection, genetic algorithms are employed in Q-Attack [27] to evade detection methods.

3.3 Methodology

In this section, we first formalize the problem of adversarial attacks on GVFL and provide a clear definition. Next, we elaborate on the threat model that underlies adversarial attacks on GVFL, outlining the assumptions and constraints of the attack scenario. Following that, we introduce our proposed method, providing insights into its framework, implementation details, and the rationale behind the algorithm.

3.3.1 Problem Definition

Definition 3.1 (*Adversarial Attack on GNN Model*) In the task of node classification, the objective of an attacker is to manipulate the behavior of the target GNN model in order to produce a desired label output for a specific adversarial example. This adversarial example is deliberately crafted by introducing imperceptible perturbations, such as rewiring edges or inserting fake nodes, to a benign input.

As expected, fed by the adversarial example, the GNN model will extract low-quality node embeddings, leading to the wrong prediction. Specifically, $G = (V, E)$ represents a graph and $f_\theta(\cdot)$ represents a GNN model. Denote A and X as the adjacency matrix of G and node features, respectively. The adversarial attack on GNN model can be formalized as

$$\begin{aligned} & \max \sum_{t \in T} L_{atk}(\text{softmax}(f_{\theta^*}(\hat{A}, \hat{X}, v_t)), y_t) \\ s.t. \quad & \theta^* = \arg \min_{\theta} \sum_{v_l \in V_L} L_{train}(f_\theta(A, X, v_l), y_l) \end{aligned} \tag{3.1}$$

where \hat{A} represents the perturbed adjacency, \hat{X} expresses the perturbed node features. v_t represents the target node and y_t is the ground truth of v_t . The parameters of GNN model θ are trained with labeled node sets V_L , resulting θ^* . It is important to note that in adversarial attacks on GNN models, the attacker has the capability to manipulate both edges and node features in order to craft adversarial examples. However, studies have shown that modifying edges tend to be more effective in influencing the model's behavior compared to modifying node features [22]. In light of these findings, this

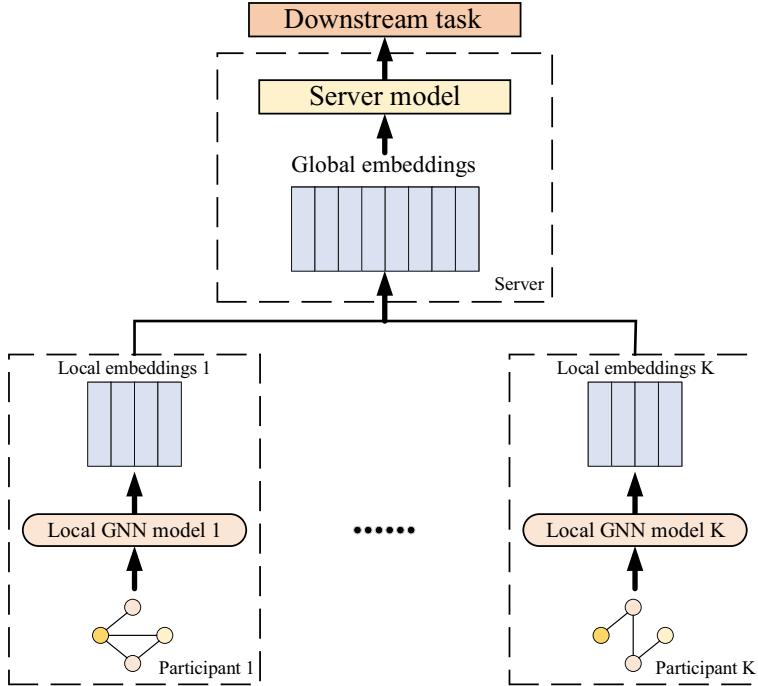


Fig. 3.1 Forward propagation of GVFL. The server model concatenates the local embeddings as global embeddings and completes the downstream task

work focuses specifically on perturbations that involve adding or deleting edges as the primary method of manipulation.

Definition 3.2 (*Adversarial Attack on GVFL*) In GVFL, each client trains a local GNN model using its private data and then aggregates the updated embeddings with the server. The objective of an attacker in this scenario is to disrupt the local embeddings generated by the clients, ultimately manipulating the server model to produce incorrect predictions. Figure 3.1 is a schematic diagram of the forward propagation for GVFL.

In this chapter, concatenating the local node embeddings is used as the combination strategy:

$$h_{global} = h_1 || h_2 || \dots || h_K \quad (3.2)$$

where K denotes the number of participants, and $||$ is the concatenating operator.

For a node classification task, the server model \mathcal{S} is a classifier to make prediction by

$$Y' = softmax(W_l \cdot \rho(\dots \rho(W_0 \cdot h_{global}))) \quad (3.3)$$

where $\{W_0 \dots W_l\}$ are weight matrices of the classifier. $\rho(\cdot)$ represents activation function such as ReLU.

Further, the GVFL uses the cross-entropy error over all labeled examples, which can be learned by gradient descent:

$$L_{train} = - \sum_{l=1}^{|V_L|} \sum_{n=1}^{|F|} Y_{ln} \ln(Y'_{ln}) \quad (3.4)$$

where V_L is the set of nodes with labels, Y_{ln} is the ground truth, and $|F|$ is the number of node classes in the graph G .

Then, the adversarial attack on GVFL is defined as

$$\begin{aligned} \hat{Y}'_t &= \text{softmax} \left(W_l \cdot \rho(\dots \rho(W_0^{v_t} \cdot (h_1^{v_t} || h_2^{v_t} || \dots || \hat{h}_m^{v_t} || h_K^{v_t}))) \right) \\ \text{s.t. } \hat{h}_m^{v_t} &= f_{\theta^*}(\hat{A}, \hat{X}, v_t) \end{aligned} \quad (3.5)$$

where $\hat{h}_m^{v_t}$ is the perturbed node embeddings of the target node v_t , which is uploaded by a malicious participant. It means a malicious participant can add perturbations to the graph by manipulating A or X . Then, the local GNN model may be confused, which will produce low-quality or even targeted node embeddings and upload the perturbed node embeddings to the server model. Therefore, the server model will make a wrong decision.

The target loss function L_{atk} of target node v_t can be defined as

$$L_{atk} = - \sum_{n=1}^{|F|} Y_{tn} \ln(Y'_{tn}) \quad (3.6)$$

which measures the difference between the prediction and the ground truth. It is easy to find the prediction of the model is worse with a larger value of L_{atk} .

3.3.2 Threat Model

Scenario. In the context of GVFL, the framework involves multiple participants who collaboratively train a central server for the task of node classification. Once the server model has been adequately trained, its predictions are shared among all participants. It's important to note that the server model is considered to be semi-honest, meaning that it behaves in a trustworthy manner by serving queries from all participants without revealing its internal parameters.

Knowledge of The Malicious Participant. In the given scenario, the malicious participant has knowledge of the structure of the server model and can only manipulate their own data. This means that the participant can attempt to attack the server model by introducing manipulations or perturbations exclusively to their own data.

However, it is important to note that the malicious participant has limited access to the server model. The server model merely serves as an API, providing probability outputs for each query made by the participant. This restricted access implies that the malicious participant is unable to obtain detailed information such as the server model's parameters or gradient information.

The Goal of The Malicious Participant. In this scenario, the objective of the malicious participant is to attack the server model by uploading perturbed embeddings. By doing so, they aim to manipulate the predictions made by the server model, causing the other benign participants, who rely on the server model for decision-making, to receive unexpected and potentially incorrect predictions for specific target examples.

3.3.3 Our Method

As mentioned in Sect. 3.3.2, the malicious attacker has the ability to manipulate their own edges or node features during testing in order to deceive the global Graph Variational Federated Learning (GVFL) system. In the aggregation process, modifying the edges impacts all dimensions of features associated with the target node, while modifying the node features only affects a subset of features. Therefore, our method is specifically designed to attack the server model by manipulating the edges of the local graph. This attack is carried out in three stages: (1) stealing node embeddings of other participants; (2) adding noises into embeddings via constructing the shadow server model; and (3) generating adversarial attacks to mislead the global GVFL. The framework of our method is illustrated in Fig. 3.2, showcasing the three

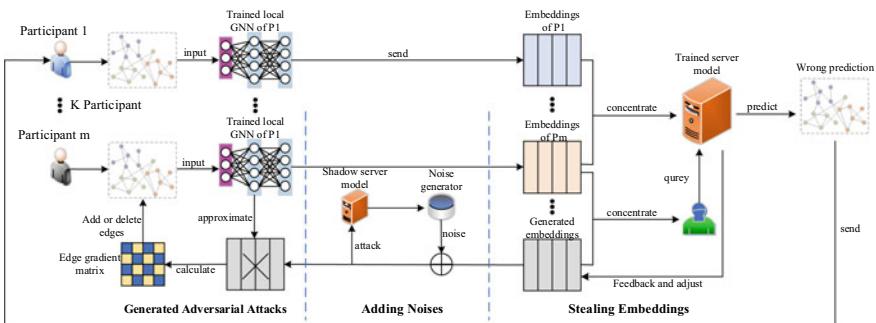


Fig. 3.2 The framework of our method. One participant is randomly selected as the malicious participant from all. The adversarial attack is conducted through three stages: (1) stealing embeddings: a global node embedding is generated, which is adjusted by querying the result of trained server model; (2) adding noises: noises are added into the generated embeddings to attack the shadow model of the server; and (3) generating adversarial attacks: guided by noise-added node embeddings of malicious participant, adversarial attacks are generated by using gradient of pairwise node. As a result, the wrong prediction of the target node will be produced by the trained server model

stages involved in the attack. The details and rationale behind each stage are further explained in this subsection, highlighting the approach taken by the attacker to compromise the global GVFL system.

3.3.3.1 Embeddings Stealing Strategy

Our approach, similar to the technique described in Luo et al. [14], leverages a generative regression network (GRN) to pilfer embeddings uploaded by other participants. First, a $(K - 1) \times d$ random noise vector is generated, where K is the number of participants in GVFL, and d is dimensions of local node embeddings. Then, a L -layers multilayer perceptron (MLP) is used to generate the target node embeddings, i.e., the embeddings uploaded by other participants. In this chapter, a three-layer MLP with ReLU is applied as the generator. The dimension of an input is $K \times d$, the dimension of an output is $(K - 1) \times d$, and the hidden units are 512 and 256, respectively. Subsequently, we concatenate the local node embeddings of the malicious participant and the target node embeddings, creating a concatenated vector denoted as h_{fake} . This concatenated vector serves as the input to the discriminator. We can obtain the classification probabilities of an adversarial example by querying the fixed parameters of the trained server model. It is important to note that the parameters of the server model remain fixed throughout this process, ensuring consistency and stability. The objective of stealing privacy can be framed as a regression problem. As a result, we employ the mean square error (MSE) as the loss function for the training process. The MSE loss function helps minimize the discrepancy between the generated target node embeddings and the actual embeddings uploaded by other participants. The formulation of the MSE loss function is as follows:

$$L_{GRN} = \frac{1}{N \times |F|} \sum_{i=1}^N \sum_{j=1}^{|F|} (\tilde{P}_{i,j} - P_{i,j})^2 \quad (3.7)$$

where N is the number of nodes, $\tilde{P} = \mathcal{S}(h_{fake})$ is the probabilities returned when node embeddings of an adversarial example are fed. P is the probabilities returned at the end of GVFL's training process.

To provide a more detailed understanding of the GRN, we simplify the model structure by using a single-layer generator model and a discriminator model S with fixed parameters. In this simplified version, we can represent the ground-truth prediction output of the discriminator model for the real node embeddings h_{real} as follows:

$$P = \mathcal{S}(h_{real}) = softmax(C_d^0 h_{real} + R_d^0) \quad (3.8)$$

Due to the parameters of S are fixed, C_d^0 is used to represent the weights of S . R_d^0 is bias of S . For a fake node embeddings h_{fake} , the output of S is

$$\begin{aligned}
P_1 &= \mathcal{S}(h_{fake}) \\
&= \mathcal{S}(g(n_{in}) || h_m) \\
&= \text{softmax}(((n_{in} || h_m)w_0 + b_0) || h_m)C_d^0 + R_d^0 \\
s.t. \quad g(n_{in}) &= (n_{in} || h_m)w_0 + b_0
\end{aligned} \tag{3.9}$$

where n_{in} is a $(K - 1) \times d$ dimension noise. h_m is local node embeddings of the malicious participant, $g(\cdot)$ is the generator, w_0 is the weights of the generator model and b_0 is bias. GRN backpropagates the loss and updates the parameters of the MLP model.

3.3.3.2 Noise Addition via Stolen Embeddings

In order to effectively degrade the performance of the server model, it is crucial to generate adversarial embeddings that cross the decision boundary and mislead the server model. One efficient method to achieve this is by using the gradient information of the server model to construct adversarial examples, which has been widely employed in attacking Graph Neural Networks (GNNs). However, in the context of GVFL, attackers do not have access to gradient information, but only the returned probabilities. Therefore, it becomes necessary to establish a shadow model locally to simulate the behavior of the server model. By stealing the global node embeddings, we can successfully create the shadow model $\tilde{\mathcal{S}}$ based on the essential knowledge, such as the training data h_{fake} , the model structure, and the target probabilities P . As mentioned in Sect. 3.3.2, the model structure and P are known. The goal is for the shadow model to output similar probabilities to the server model when provided with the same examples. To achieve this, we employ the Mean Square Error (MSE) as the training target for the shadow model. This ensures that the output probabilities of the shadow model closely match the target probabilities P . The MSE formulation can be expressed as follows:

$$L_{shadow} = \frac{1}{N \times |F|} \sum_{i=1}^N \sum_{j=1}^{|F|} (\tilde{\mathcal{S}}(h_{fake}) - P_{i,j})^2 \tag{3.10}$$

where $\tilde{\mathcal{S}}(h_{fake})$ is the output of $\tilde{\mathcal{S}}$.

Then, fast gradient sign method (FGSM) [28] is adopted as noise generator by using the gradient information of the shadow model $\tilde{\mathcal{S}}$.

For a target node v_t , its noise-added embeddings $\bar{h}_{fake}^{v_t}$ could be represented as

$$\bar{h}_{fake}^{v_t} = h_{fake}^{v_t} + \epsilon \cdot \text{sign}\left(\frac{\partial L_{atk}}{\partial h_{fake}^{v_t}}\right) \tag{3.11}$$

where $\text{sign}(\cdot)$ is gradient's direction, $\epsilon \in [0, 1]$ is noise scale.

In particular, the reason for using MSE as the target to establish the shadow model is as follows. Suppose using a mapping equation to represent the server model:

$$\hat{k}_{\mathcal{S}}(h) = \arg \max_k \mathcal{S}(h) \quad (3.12)$$

where h is the node embeddings and k is k -th class of \mathcal{S} . The classification boundary function can be expressed as

$$\mathcal{S}_k(h) - \hat{k}_{\mathcal{S}}(h) = 0 \quad (3.13)$$

By applying the first-order Taylor expansion, the approximate classification boundary function of \mathcal{S} is denoted as $\tilde{\mathcal{S}}$

$$\begin{aligned} B_{\mathcal{S}} : & \mathcal{S}_k(h_{real}) + h_{real} \nabla \mathcal{S}_k(h_{real}) \\ & - \hat{k}_{\mathcal{S}}(h_{real}) - h_{real} \nabla \hat{k}_{\mathcal{S}}(h_{real}) = 0 \end{aligned} \quad (3.14)$$

$$\begin{aligned} B_{\tilde{\mathcal{S}}} : & \tilde{\mathcal{S}}_k(h_{fake}) + h_{fake} \nabla \tilde{\mathcal{S}}_k(h_{fake}) \\ & - \hat{k}_{\tilde{\mathcal{S}}}(h_{fake}) - h_{fake} \nabla \hat{k}_{\tilde{\mathcal{S}}}(h_{fake}) = 0 \end{aligned} \quad (3.15)$$

The MSE of the shadow model is aiming to approximate the classification boundary of $\tilde{\mathcal{S}}$ and \mathcal{S} . Thus, the target can be converted to

$$\begin{aligned} & \arg \min_{\tilde{\mathcal{S}}} MSE(\tilde{\mathcal{S}}(h_{fake}), \mathcal{S}(h_{real})) \\ \Leftrightarrow & \arg \min_{\tilde{\mathcal{S}}} |B_{\tilde{\mathcal{S}}} - B_{\mathcal{S}}| \\ \Leftrightarrow & \arg \min_{\tilde{\mathcal{S}}} |\tilde{\mathcal{S}}(h_{fake}) - \mathcal{S}(h_{real})| \\ & + |(h_{fake} - h_{real})(\nabla \tilde{\mathcal{S}}(h_{fake}) - \nabla \mathcal{S}(h_{real}))| \end{aligned} \quad (3.16)$$

Then, assuming $\nabla \tilde{\mathcal{S}}(h_{fake}) \approx \nabla \mathcal{S}(h_{real})$, and the noise generated by FGSM against the shadow model can also attack the server model in GVFL successfully since the approximate direction of the gradient is obtained.

3.3.3.3 Adversarial Attack via Noise-Added Embeddings

Following the aforementioned steps, we obtain the noise-added embeddings $\bar{h}^{v_t}_{fake}$. We then proceed to extract the portion of the noise-added embeddings $\bar{h}^{v_t} m$ that corresponds to the malicious participant from $\bar{h}^{v_t} fake$. Assuming that $\bar{h}^{v_t} fake$ can successfully confuse the shadow model, the remaining task is to modify certain edges in the original graph to make the node embeddings approximate $\bar{h}^{v_t} m$. In

order to achieve this, our method focuses on making the target node's embeddings and $\bar{h}_m^{v_t}$ more similar by introducing perturbations into the adjacency matrix A . This perturbation process aims to align the embeddings and enhance the resemblance between them.

$$\arg \min_{\hat{A}} MSE(f_{\theta^*}(\hat{A}, X, v_t), \bar{h}_m^{v_t}) \quad (3.17)$$

where $\| \hat{A} - A \|_0$ is twice the budget Δ , i.e., $\| \hat{A} - A \|_0 \leq 2\Delta$ due to the symmetry of the adjacency matrix.

To expedite the search for suitable adversarial edges, we leverage the gradient of pairwise nodes, inspired by the work of Chen et al. [21]. This involves computing the edge gradient matrix, which can be expressed as follows:

$$\begin{aligned} g_{u,v} &= \frac{\partial L_t}{\partial A_{u,v}} \\ s.t. \quad L_t &= \frac{1}{d} \sum_{i=1}^d ([f_{\theta^*}(\hat{A}, X, v_t)]_i - [\bar{h}_{fake}^{v_t}]_i)^2 \end{aligned} \quad (3.18)$$

where (u, v) is any pairwise node in the graph G , $[\cdot]_i$ denotes the i -th element of node embeddings and d is the dimensions of local node embeddings. Considering the symmetry of the adjacency matrix of the undirected graphs, the gradient matrix is symmetric as:

$$g_{sym} = \frac{g + g^T}{2} \quad (3.19)$$

where $.^T$ is the transpose symbol. The adversarial edge e_{adv} is selected by

$$e_{adv} = (u, v) \leftarrow \arg \max_{u, v} -g_{sym} \quad (3.20)$$

In order to minimize the discrepancy between the output of the local GNN model and the noise-added embeddings, instead of maximizing L_{atk} , it is necessary to invert the gradient values. This inversion allows us to identify the most influential edges for achieving the desired similarity between the embeddings. Finally, the adversarial edges are added iteratively into the clean adjacency matrix within a specified budget Δ :

$$\hat{A}_{u,v} = -A_{u,v} + 1 \quad (3.21)$$

where (u, v) comes from e_{adv} and \hat{A} is symmetrical.

3.3.4 Complexity Analysis

The time complexity of our method for generating adversarial attacks can be divided into four parts: the training time of the GRN (T_{GRN}), the training time of the shadow model (T_{shadow}), the time taken for adding noise, and the time taken for selecting adversarial edges. Therefore, the overall time complexity of our method can be expressed as

$$\mathcal{O}(T_{GRN}) + \mathcal{O}(T_{shadow}) + \mathcal{O}(1) + \mathcal{O}(\Delta) \sim \mathcal{O}(N) \quad (3.22)$$

where Δ is the attack budget. $\mathcal{O}(T_{GRN})$ and $\mathcal{O}(T_{shadow})$ are the complexity of the training time of GRN and the shadow, depending on maximum training epochs T_{GRN} and T_{shadow} , respectively. $\mathcal{O}(1)$ indicates the complexity of noise adding time because FGSM is a one-step noise generator. $\mathcal{O}(\Delta)$ is the complexity of adversarial edges selecting time, which is controlled by the attack budget Δ . $\mathcal{O}(N)$ indicates the complexity of our method is linear, according to the total number of the above steps N .

The parameters of our method include GRN’s parameters and shadow model’s parameters. Therefore, the space complexity is

$$\begin{aligned} & \mathcal{O}(K \times d \times n_0 + n_0 \times n_1 + \cdots + \\ & (K - 1) \times d \times n_{L-1}) + \mathcal{O}(\tilde{S}) \sim \mathcal{O}(M^2) \end{aligned} \quad (3.23)$$

where K is the number of participants, d is the dimensions of local node embeddings, $[n_0, \dots, n_{L-1}]$ is the number of the units in GRN models and \tilde{S} is the shadow model. $\mathcal{O}(M^2)$ indicates that the space complexity depends on the memory occupied by the model’s weight matrix, whose is squared-level.

3.4 Experiments

To thoroughly evaluate the effectiveness of our method, we conduct experiments to assess both the dual-participant-based GVFL and the multi-participant-based GVFL in terms of their attack performance.

3.4.1 Datasets

Four graph-structured datasets are used to evaluate the performance of the proposed method, including Cora [29], Cora_ML [29], Citeseer [29], and Pol.Blogs [30].

In GVFL, the dataset is divided among different clients. To ensure that the clients have non-overlapping edges, we assume that each client is given a separate subset of

the dataset. This leads to the emergence of isolated nodes in the graph data of each client, which can hinder collaborative training of the server model. To address this issue, we employ different splitting strategies based on the number of clients involved. For dual-participant scenarios, both the edges and node features of the dataset are evenly divided into two parts, resulting in a small portion of isolated nodes appearing in each partition. On the other hand, for multi-participant scenarios, the node features are evenly divided among the participants while retaining the complete topology of the graph. Since the segmentation process is random, we repeat the dataset division and testing procedure 10 times. This allows us to record the average accuracy of the trained server model, providing a comprehensive evaluation of its performance.

3.4.2 Local GNN Model

To demonstrate that our method is effective in various GNN structure-based GVFLs, we adopt three GNN models as local participants.

- **Graph Convolutional Network (GCN)** [31]: The proposed method utilizes a layer-wise propagation rule that is based on a first-order approximation of spectral convolutions on graphs. Specifically, for node classification tasks, a two-layer GCN is employed as the local GNN model within the GVFL framework. The node is represented as

$$h_i^{(l+1)} = \rho \left(\sum_{j \in ne(i)} \frac{1}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} h_j^{(l)} W^{(l)} \right) \quad (3.24)$$

where $h^{(l+1)}$ is $(l + 1)$ -th layer's node representation and $W^{(l)}$ is the parameters of l -th layer. Node j belongs to neighbor node set ne of node i . $\tilde{D}_{ii} = \sum_j (A + I_N)$ is the degree matrix of $A + I_N$, and I_N is the identity matrix.

- **Simple Graph Convolution (SGC)** [32]: The approach we use is a linearized version of the GCN that does not include an activation function. In this linearized GCN, the aggregation function can be formulated as

$$h_i^{(l+1)} = \sum_{j \in ne(i)} \frac{1}{\sqrt{\tilde{D}_{ii} \tilde{D}_{jj}}} h_j^{(l)} W^{(l)} \quad (3.25)$$

SGC usually outperforms GCN for node classification.

3.4.3 Attack Baselines

As this is the first attempt to perform adversarial attacks on GVFL, we select three attack methods from the centralized setting as our baseline approaches. To adapt these attack methods to GVFL, we utilize the probabilities returned by the server model and the local data to train a surrogate model, which is essentially the same as the local GNN model. The baselines are briefly described as follows.

- **RND** [19]: we assume that the connection of different types of nodes will affect the prediction result. For a given target node, RND randomly samples a node whose predicted label is unequal to the target node. Then, add an edge between the target node and the sampled node.
- **NETTACK** [19]: it generates the adversarial edges guided by two evaluation functions from selected candidate edges and features. It modifies the highest scoring edges or features and updates the adversarial network iteratively to confuse GNNs.
- **FGA** [21]: it constructs the edge gradient network based on original network firstly. Then, it generates the adversarial edges iteratively with the maximal absolute edge gradient til it reaches the attack budget Δ .

3.4.4 Experiment Setup

In each local GNN model, we employ a two-layer GNN model to extract local node embeddings. The dimension of the embeddings is set to 16, while the number of hidden units is fixed at 32. For the GCN and the SGC, the ReLU activation function is used. The GVFL framework is trained for 200 epochs using the Adam optimizer with a learning rate of 0.01. Considering the sparsity of the dataset and the need for attack concealment, the attack budget Δ is set to a fixed value of 1. The main evaluation metric used is the accuracy of the server model, where a lower accuracy indicates a better performance for the attacker.

3.4.5 Attack Performance

In this subsection, our method is conducted on three real-world datasets in two main scenes, i.e., dual-participant-based GVFL and multi-participant-based GVFL.

3.4.5.1 Attack on Dual-Participant-Based GVFL

In the described setting, the datasets are randomly divided into two subsets, considering both the edges and node features. To evaluate the performance of the attackers

comprehensively, multi-perspective metrics are employed, including the node classification accuracy, precision, recall, F1-Score, mean absolute error (MAE), and Log Loss.

To examine the generalizability of our method across different graph segmentation scenarios, we conduct a total of 10 experiments and report the average accuracy along with the standard deviation. The best attack performance is highlighted in bold. In Fig. 3.3, we present five additional metrics to evaluate the proposed method from various perspectives. The precision, recall, and F1-Score are employed to measure the performance of the GVFL framework. A lower score from the attacker indicates a better attack performance. On the other hand, MAE and Log Loss metrics are utilized to assess the level of confusion caused to the target server model. A higher value in these metrics signifies a better attack performance.

- Our method achieves the SOTA attack performance compared with baselines. As an example, our method exhibits a significant degradation in accuracy for the baselines, surpassing a 10% decrease in performance on datasets such as Cora, Cora_ML, and Citeseer. This outcome demonstrates the superior performance of our approach compared to the baselines. Furthermore, it is worth noting that the standard deviation reveals the stability of our method, which is generally superior to that of NETTACK and FGA. In Fig. 3.3, we present the results for the other five metrics, where our method outperforms the baselines in terms of attack performance for each metric.

		Cora			Cora ML			Citeseer								
		Precision	Recall	F1-Score	MAE	Log Loss	Precision	Recall	F1-Score	MAE	Log Loss	Precision	Recall	F1-Score	MAE	Log Loss
GCN	Clean	0.68	0.73	0.69	0.10	1.00	0.80	0.78	0.79	0.07	0.65	0.64	0.64	0.63	0.15	1.14
	RND	0.62	0.66	0.63	0.12	1.09	0.74	0.72	0.73	0.09	0.76	0.58	0.59	0.58	0.17	1.21
	NETTACK	0.54	0.47	0.49	0.15	1.75	0.59	0.55	0.54	0.14	1.79	0.48	0.44	0.45	0.19	1.68
	FGA	0.55	0.57	0.56	0.14	1.25	0.62	0.59	0.59	0.12	1.19	0.55	0.53	0.53	0.18	1.26
SGC	Ours	0.44	0.39	0.41	0.18	2.09	0.46	0.43	0.43	0.18	2.19	0.41	0.32	0.33	0.23	2.15
	Clean	0.70	0.72	0.70	0.10	0.95	0.81	0.78	0.80	0.07	0.65	0.64	0.64	0.64	0.14	1.11
	RND	0.65	0.68	0.66	0.12	1.00	0.75	0.72	0.73	0.09	0.75	0.59	0.60	0.59	0.16	1.18
	NETTACK	0.58	0.58	0.57	0.14	1.30	0.60	0.57	0.56	0.14	1.72	0.49	0.46	0.47	0.19	1.57
SGC	FGA	0.57	0.55	0.55	0.14	1.26	0.63	0.58	0.58	0.12	1.19	0.55	0.53	0.53	0.18	1.23
	Ours	0.47	0.46	0.45	0.17	1.66	0.48	0.45	0.46	0.17	2.01	0.48	0.39	0.40	0.21	1.86

Fig. 3.3 Multi-perspective metrics to measure the performance of attacks. For precision, recall, and F1-Score, the lower the attacker's score, the better the attack performance is, and for MAE and Log Loss, the higher the attacker's score, the better the attack performance is

- *The integrity of graph-structured data affects the performance of the model.* Compared to the performance of centralized GNN models, the performance of the GVFL framework shows a decrease. This can be attributed to the fact that the local GNN models are unable to capture complete node information when edges are assigned to different participants, resulting in a weakening of the quality of the local node embeddings. Moreover, GVFL exhibits better performance on graphs with higher degrees, such as a drop of 8.7% in performance for GVFL based on GCN on the Cora dataset (with an average degree of 2.00). Similar results can be observed across different instances of GVFL. This phenomenon can be interpreted as denser graphs retaining more relational information even after data segmentation, thereby enabling GVFL to perform better in such cases.

In order to conduct a more detailed comparison of the performance of the attack methods, we employ the classification margin as a metric to evaluate the influence of these methods on the probabilities of the model’s predictions, following the approach proposed by Zügner et al. [19]. Specifically, we randomly select 100 nodes from the test set that are correctly classified by the model. These nodes satisfy:

- the 20 nodes with the highest classification margin,
- the 20 nodes with the lowest classification margin but still be classified correctly, and
- the 60 nodes are selected randomly.

On datasets like Cora, Citeseer, and Pubmed, over half of the target nodes have a classification margin below 0 when attacked using our method. This indicates a higher attack performance on these datasets compared to NETTACK and FGA. Furthermore, our method outperforms NETTACK and FGA in terms of the number of nodes with low classification margin on the Cora_ML dataset. In the case of RND, only a few nodes achieve a low classification margin, which explains its inferior attack performance. Additionally, there are still some nodes with higher classification margin, indicating that our method may not be able to successfully attack these nodes. This could be attributed to the limitations of the single-edge attack, which might not be sufficient to cause misclassification in these nodes.

The experiments conducted above highlight the vulnerability of the GVFL framework to adversarial attacks. Although the existing adversarial attacks that can be transferred to GVFL may not be powerful enough in the absence of knowledge about GVFL, GVFL still exhibits susceptibility to attacks. This vulnerability can be attributed to the GNN model employed in GVFL. Adversarial attacks manipulate the GNN model’s extraction of node embedding vectors, leading to incorrect embeddings and ultimately confusing the server model. Furthermore, our method takes advantage of the leakage of global node embeddings in GVFL to establish a shadow server model that guides the attack towards success. By leveraging additional knowledge of GVFL, our method demonstrates superior attack performance.

3.4.5.2 Attack on Multi-Participant-Based GVFL

In order to assess the effectiveness of our method in an FL setting with multiple participants, we conducted experiments with various numbers of participants ranging from 2 to 6. To ensure that there are no isolated nodes in the dataset, we randomly distributed node features to each participant without segmenting the edges. However, for the Pol.Blogs dataset, which lacks node features, the experiments were conducted on the remaining three datasets. The results are shown in Table 3.1.

In this setting, the performance of GVFL is closer to centralized GNN models compared to edge segmentation, as it retains the complete relationships between node pairs. However, as the number of participants increases, the accuracy of GVFL gradually decreases due to information loss in node feature segmentation. Similar to the edge segmentation setting, our method achieves optimal attack performance with two participants. Generally, the performance of attack methods decreases as the number of participants increases. This is because the server model treats each participant equally, and as the number of participants grows, the contribution of the malicious participant's node embeddings decreases. As a result, it becomes more challenging to confuse the server model. For instance, on the Pubmed dataset, all attack methods practically fail with single-edge attacks when the number of participants is six. Interestingly, this pattern is not consistent across all experiments. For example, in the Citeseer dataset, the accuracy of GVFL decreases when the number of participants is six. The contribution of each participant, denoted as C_i , is defined as $C_i = \frac{acc_i}{\sum_j^K acc_j = 1}$, where K is the number of participants and acc_i is the accuracy of the server model

Table 3.1 Attack on multi-participant-based GVFL. Each row represents experiments on different numbers of participants

Datasets	Attack	2P	3P	4P	5P	6P
Cora	Clean	0.798	0.773	0.782	0.754	0.765
	RND	0.768	0.756	0.759	0.746	0.760
	NETTACK	0.728	0.730	0.688	0.697	0.746
	FGA	0.694	0.752	0.722	0.725	0.746
	Ours	0.608	0.706	0.635	0.650	0.734
Cora_ML	Clean	0.842	0.845	0.837	0.840	0.841
	RND	0.821	0.821	0.819	0.833	0.822
	NETTACK	0.696	0.691	0.741	0.823	0.824
	FGA	0.749	0.711	0.800	0.820	0.818
	Ours	0.618	0.608	0.667	0.743	0.775
Citeseer	Clean	0.662	0.645	0.638	0.639	0.627
	RND	0.617	0.637	0.630	0.646	0.635
	NETTACK	0.531	0.532	0.558	0.634	0.548
	FGA	0.526	0.540	0.561	0.634	0.588
	Ours	0.466	0.473	0.488	0.617	0.453

when only participant i 's node embeddings are used as input. Observing the results on the Cora_ML dataset, we can see that as the number of participants increases, the contribution of each participant gradually decreases, making the server model less susceptible to attacks. However, on Citeseer, when the number of participants is six, the proportion of the malicious participant's contribution increases, leading to successful attacks by our method. We attribute this to the random distribution of node features in this setting, where not all participants receive useful node features. As a result, a small perturbation becomes powerful enough to fool the server model. Conversely, when benign participants receive key node features, the slight structural perturbation from the malicious participants is not sufficient to impact the server model's performance. In other words, the model becomes biased towards the data. Additionally, the properties of the datasets also impact the attack performance. For instance, on the Citeseer dataset, which is the most sparse, our method maintains a good attack performance. The sparseness of the dataset enhances the influence of structural attacks on the perturbations of local node embeddings.

To summarize, in the multi-participant setting, the performance of attackers generally decreases as the number of participants increases. This trend can be attributed to the diminishing influence of each participant's server model when more participants are involved. Additionally, the success of adversarial attacks may be attributed to the model's bias towards the data. This indicates that if the server model relies heavily on high-contributing participants, it becomes easier for attacks initiated by these participants to deceive the server model. Therefore, establishing a fair evaluation mechanism for participants could potentially enhance the robustness of GVFL against adversarial attacks. This mechanism would ensure that the server model does not disproportionately rely on certain participants, reducing the vulnerability to targeted attacks. Overall, these insights point towards future research directions for improving the robustness of GVFL in multi-participant scenarios and addressing potential biases in the server model's decision-making process.

3.5 Conclusion

We propose a novel adversarial attack method which is specifically designed for the GVFL framework. By leveraging the privacy leakage present in GVFL and utilizing the gradient of pairwise nodes, our method achieves state-of-the-art attack performance when compared to existing baseline methods. Extensive experiments have been conducted to evaluate the effectiveness and efficiency of our method. The results demonstrate the superior performance of our method in compromising the accuracy and robustness of GVFL. These experiments not only validate the effectiveness of our approach but also shed light on the inherent vulnerabilities of GVFL, even when defensive measures are implemented.

References

1. McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging (2016). [arXiv:1602.05629](https://arxiv.org/abs/1602.05629)
2. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol. (TIST)* **10**(2), 1–19 (2019)
3. Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., Beaufays, F.: Applied federated learning: improving google keyboard query suggestions (2018). [arXiv:1812.02903](https://arxiv.org/abs/1812.02903)
4. Ge, S., Wu, F., Wu, C., Qi, T., Huang, Y., Xie, X.: Fedner: medical named entity recognition with federated learning (2020). [arXiv:2003.09288](https://arxiv.org/abs/2003.09288)
5. Long, G., Tan, Y., Jiang, J., Zhang, C.: Federated learning for open banking. In: Federated Learning, pp. 240–254. Springer, Berlin (2020)
6. Konečny, J., McMahan, H.B., Ramage, D., Richtárik, P.: Federated optimization: distributed machine learning for on-device intelligence (2016). [arXiv:1610.02527](https://arxiv.org/abs/1610.02527)
7. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
8. Mohri, M., Sivek, G., Suresh, A.T.: Agnostic federated learning. In: International Conference on Machine Learning, pp. 4615–4625. PMLR (2019)
9. Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., Khazaeni, Y.: Bayesian nonparametric federated learning of neural networks. In: International Conference on Machine Learning, pp. 7252–7261. PMLR (2019)
10. Mugunthan, V., Peraire-Bueno, A., Kagal, L.: Privacyfl: A simulator for privacy-preserving and secure federated learning. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 3085–3092 (2020)
11. Hitaj, B., Ateniese, G., Perez-Cruz, F.: Deep models under the gan: information leakage from collaborative deep learning. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 603–618 (2017)
12. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE symposium on security and privacy (SP), pp. 739–753. IEEE (2019)
13. Zhu, L., Han, S.: Deep leakage from gradients. In: Federated Learning, pp. 17–31. Springer, Berlin (2020)
14. Luo, X., Wu, Y., Xiao, X., Ooi, B.C.: Feature inference attack on model predictions in vertical federated learning (2020). [arXiv:2010.10152](https://arxiv.org/abs/2010.10152)
15. Zhang, S., Xiang, L., Yu, X., Chu, P., Chen, Y., Cen, C., Wang, L.: Privacy-preserving federated learning on partitioned attributes (2021). [arXiv:2104.14383](https://arxiv.org/abs/2104.14383)
16. Li, O., Sun, J., Yang, X., Gao, W., Zhang, H., Xie, J., Smith, V., Wang, C.: Label leakage and protection in two-party split learning (2021). [arXiv:2102.08504](https://arxiv.org/abs/2102.08504)
17. Iwendi, C., Rehman, S.U., Javed, A.R., Khan, S., Srivastava, G.: Sustainable security for the internet of things using artificial intelligence architectures. *ACM Trans. Internet Technol. (TOIT)* **21**(3), 1–22 (2021)
18. Ahmed, U., Lin, J.C.W., Srivastava, G.: Generative ensemble learning for mitigating adversarial malware detection in iot. In: 2021 IEEE 29th International Conference on Network Protocols (ICNP), pp. 1–5. IEEE (2021)
19. Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial attacks on neural networks for graph data. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2847–2856 (2018)
20. Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L.: Adversarial attack on graph structured data (2018). [arXiv:1806.02371](https://arxiv.org/abs/1806.02371)
21. Chen, J., Wu, Y., Xu, X., Chen, Y., Zheng, H., Xuan, Q.: Fast gradient attack on network embedding (2018). [arXiv:1809.02797](https://arxiv.org/abs/1809.02797)
22. Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., Zhu, L.: Adversarial examples on graph data: deep insights into attack and defense (2019). [arXiv:1903.01610](https://arxiv.org/abs/1903.01610)

23. Li, J., Xie, T., Liang, C., Xie, F., He, X., Zheng, Z.: Adversarial attack on large scale graph. *IEEE Trans. Knowl. Data Eng.* (2021)
24. Wang, X., Cheng, M., Eaton, J., Hsieh, C.J., Wu, F.: Attack graph convolutional networks by adding fake nodes (2018). [arXiv:1810.10751](https://arxiv.org/abs/1810.10751)
25. Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., Huang, J.: A restricted black-box adversarial framework towards attacking graph embedding models. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 3389–3396. AAAI Press (2020)
26. Ma, Y., Wang, S., Derr, T., Wu, L., Tang, J.: Attacking graph convolutional networks via rewiring (2019). [arXiv:1906.03750](https://arxiv.org/abs/1906.03750)
27. Chen, J., Chen, L., Chen, Y., Zhao, M., Yu, S., Xuan, Q., Yang, X.: Ga-based q-attack on community detection. *IEEE Trans. Comput. Soc. Syst.* **6**(3), 491–503 (2019)
28. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). [arXiv:1412.6572](https://arxiv.org/abs/1412.6572)
29. McCallum, A.K., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. *Inf. Retr.* **3**(2), 127–163 (2000)
30. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 us election: divided they blog. In: *Proceedings of the 3rd International Workshop on Link Discovery*, pp. 36–43 (2005)
31. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2016). [arXiv:1609.02907](https://arxiv.org/abs/1609.02907)
32. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: *International Conference on Machine Learning*, pp. 6861–6871. PMLR (2019)

Chapter 4

A Novel DNN Object Contour Attack on Image Recognition



4.1 Introduction

In recent years, deep neural networks (DNNs) have gained widespread adoption in various professional tasks, including computer vision [1], bioinformatics [2], and natural language processing [3]. Due to their wide applications, the security prowess of DNNs has garnered significant attention, as highlighted in [4]. This is particularly evident in the fields of autopilot [5] and disease diagnosis [6]. Recent studies have indicated that deep neural networks (DNNs) are susceptible to adversarial examples. Adversarial examples, which are perceptually similar to benign inputs but misclassified by DNNs with high confidence levels, have posed significant threats to various applications, including those in the fields of intelligent video surveillance (e.g., the road sign recognition system of autopilot [7]), autopilot, and disease diagnosis (e.g., speech recognition system [8]).

According to the threat model, adversarial attacks can be categorized into two types: white-box and black-box attacks. White-box attacks assume full access to the target model, including its structure, parameters, training process, and training data. Examples of white-box attacks include Fast Gradient Sign Method (FGSM) [9], Momentum Iterative FGSM (MI-FGSM) [10], and Projected Gradient Descent (PGD) [11]. Black-box attacks treat the target model as an oracle. Examples of black-box attacks include Zeroth-Order Optimization (ZOO) [12], Universal Perturbations for Steering to Exact Targets (UPSET) [13], and Antagonistic Network for Generating Rogue Images (ANGRI) [13]. Generally, white-box attacks tend to generate adversarial examples with smaller perturbations that are less transferable, which can be defended using strategies such as adversarial training and gradient regularization. In contrast, black-box attacks typically generate adversarial examples with larger perturbations that are more transferable than those generated by white-box attacks.

In short, white-box attacks have the advantages of fast attack speed and small disturbance, but they cannot be effectively applied to cases where the model is unknown. Black-box attacks only need to obtain the output label or confidence of the model, and can bypass most defenses, but there are obvious perturbations and high time costs.

Furthermore, a few studies have examined the effectiveness of adversarial attacks in terms of perturbation and correlation with benign examples [14, 15]; however, these studies still do not mention visual results that can intuitively help understand the correlation between perturbations and target labels. In addition, most attacks (e.g., FGSM [9], MI-FGSM [10], and PGD [11]) are based on global gradient information when adding perturbation, so that many redundant perturbation is added to the background. To solve these problems, we propose a white-box targeted attack method.

The objectives of our method primarily encompass three facets: 1. Uncover more model vulnerabilities. 2. Enhance the model’s robustness. 3. Supply a novel perspective on attacking contour and gradient correlation. Although our method is an attack method, it can improve the model’s robustness through adversarial training [16, 17]. This involves training the target model based on the adversarial examples crafted by our method. The major contributions of this research can be summarized as follows:

- In this chapter, we investigate the relationship between the performance of deep neural networks (DNNs) and the accuracy of object contour localization. Our findings reveal the existence of adversarial attacks that are triggered by their degradation towards the object contour location and the concentration of DNNs in those areas.
- Inspired by this, we introduce the attention perturbation adversarial technique to adversarial attacks and develop our method. Our method explores the relationship between perturbations and object contours to achieve superior attack performance with fewer perturbations.
- To visualize the distinction in attention areas between adversarial examples and benign ones, we employ heatmaps. These heatmaps reveal that our method concentrates perturbations near the object contour, resulting in a more targeted attack performance and improved transferability. Furthermore, extensive experiments are conducted to validate the effectiveness of our method.

4.2 Related Works

We select several leading adversarial attacks for comparison, which serve as baselines, including FGSM [9], MI-FGSM [10], PGD [11], Basic Iterative Method (BIM) [18], DeepFool [19], and Carlini and Wagner Attacks (C&W) [20].

A widely used untargeted white-box attack, namely FGSM, has been proposed by Goodfellow et al. [9]. The backward propagation gradient from the target model is applied in this attack. Dong et al. [10] introduced a momentum-based white-box attack to enhance the adversarial attack ability, which is referred to as MI-FGSM. Madry et al. [11] proposed PGD, a standard white-box attack for large-scale constrained optimization, which performs one step of gradient descent and then clips all the coordinates to be within the box. Kurakin et al. [18] demonstrated BIM,

a white-box attack based on the standard convex optimization method, which is equivalent to the projected gradient descent. Moosavi et al. [19] proposed DeepFool, a simple but effective untargeted white-box attack. Finally, Carlini and Wagner [20] designed C&W attack, a state-of-the-art white-box attack whose essence is of the same mold as a refined iterative gradient attack but with the Adam optimizer.

The white-box attacks (e.g., DeepFool [19] and C&W [20]) considered optimizing gradient information when generating adversarial examples, instead of adding perturbation along the global gradient. However, they still do not consider the correlation between the gradient and the object contour like our method, which adds perturbation near the contour.

4.3 Methodology

Adversarial examples are typically assessed based on their attack capability and perturbation scale. The attack capability is determined by the size and distribution of perturbations. Statistically, larger perturbations lead to a stronger attack capability; however, attackers create examples with minor perturbations to conceal the attack. Consequently, adversarial attacks present a paradoxical, multi-objective problem. Two questions thus arise: how to find the appropriate position to add perturbations, and what is the smallest scale of the perturbations? To address these queries, we suggest a novel approach known as our method to generate appropriate perturbations for successful adversarial attacks.

The comprehensive blueprint of our method is illustrated in Fig. 4.1, which involves the targeted DNN assault, the extraction of the shallow layer's feature map from the model, the application of bilinear upsampling to augmentation, and the magnification of perturbations by a factor of 50 for enhanced visualization. Considering the constraints of attack ability, an iterative optimization process is employed to minimize the perturbations. Additionally, an attention-based perturbation adversarial technique is employed to more effectively search for a suitable distribution of perturbations.

4.3.1 Perturbation Distribution of Adversarial Attack

Our findings indicate that the accuracy of a DNN's performance is not solely dependent on the precise localization of object contours [21], but also on the extent of the target object area. This observation is supported by the comparison of benign and adversarial examples generated by various attack strategies against the ResNet-v2 model, as shown in Fig. 4.2. The significant difference between the benign and adversarial examples, which may go unnoticed by the human eye, is clearly illustrated by the Grad-CAM maps. As shown in Fig. 4.2, it can be observed that the invisible

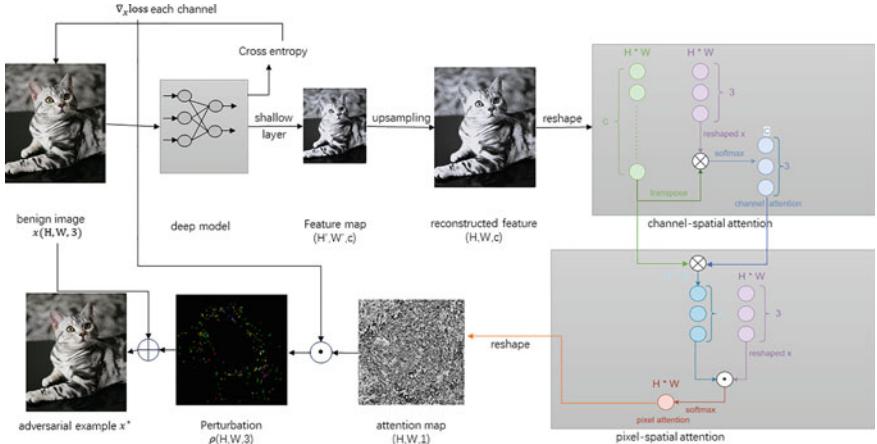


Fig. 4.1 Framework of the our method attack method. The “reconstructed feature map” is obtained by upsampling from the “feature map”. The channel-spatial attention module is used to produce the channel-spatial attention weight W_c . Subsequently, the pixel-spatial attention module is used to produce the pixel-spatial attention weights W_p , resulting in an “attention map”

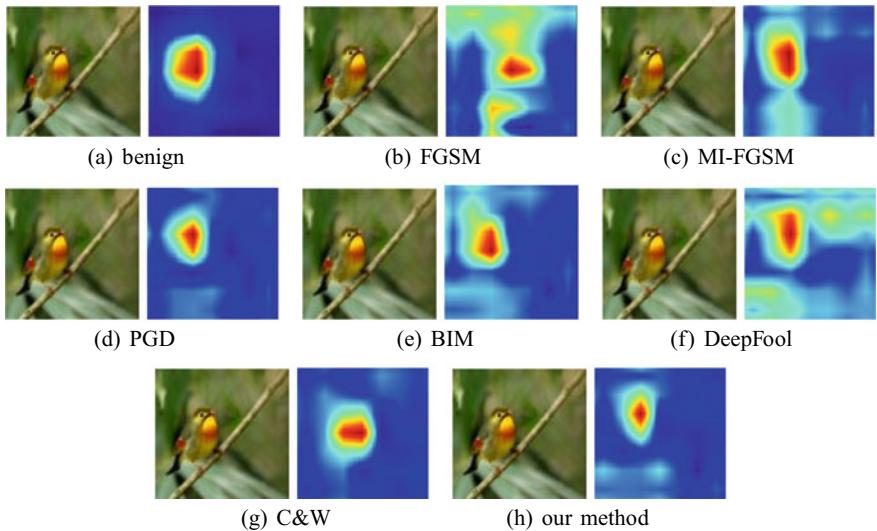


Fig. 4.2 The attention region visualization of benign and adversarial examples based on Grad-CAM [22]. In subfigure **a**, the left is benign example and the right is Grad-CAM map. From subfigure **b-h** are the adversarial examples obtained by different attacks against ResNet-v2 and their Grad-CAM maps

adversarial perturbation is magnified in the hidden layers of the DNN, ultimately impacting the DNN's perceived scope.

This is in line with the notion that the placement of the object contour within the image is a key factor in successful adversarial attacks. It can be deduced that the DNN's focus on the object contour has diminished, leading to a more restricted object region that is bound by this attention. Conversely, if the contour in the background is misplaced, it expands the range of the background. Therefore, we can devise methods to carry out attacks from two perspectives: diminishing the DNN's precision in locating the object contour or restricting the scope of the areas under scrutiny.

The gradient can be seen as a quantitative measure of the sensitivity of the input pixels to the output confidence. Considering that the convolution kernel functions as a filter, and that various convolution kernels identify different features of the same image, the background may be subsequently eliminated during forward propagation due to the absence of perceptible texture information. The gradient can be seen as a quantitative measure of the sensitivity of the input pixels to the output confidence. On the contrary, due to the rich texture features surrounding the contour of the target object, it is retained during forward propagation. This avoids gradient dispersion in backward propagation. The gradient can be seen as a quantitative measure of the sensitivity of the input pixels to the output confidence.

Our method aims to concentrate the perturbations on the classification-feature distribution that encompasses contour information. In this regard, an attention perturbation adversarial technique associated with the feature map layer is employed to capture the outlining features of an object.

4.3.2 *Spatial Attention*

Typically, there are two types of attention perturbation adversarial techniques: hard attention and soft attention [23]. Hard attention is a random process that assigns weights based on the Bernoulli distribution, while soft attention is a parametric weighting method that can be integrated into the architecture of the DNN for end-to-end training. In our method, soft attention is employed to search for imperceptible perturbations near the target object contour.

In comparison to shallow features, deep features possess a larger visual field; however, they offer significantly lesser spatial information. Consequently, we reconstruct the shallow feature in the same dimensional space as the input data by utilizing bilinear interpolation. The different channels of feature maps are corresponding to the filtering results of the various convolutional kernels. The attention perturbation adversarial technique for the perturbation distribution search is composed of channel-spatial attention and pixel-spatial attention. Feature distribution is concentrated by assigning varying weights to the channels of a feature map, while pixel regions are focused on by assigning distinct weights to the pixels of a feature map. The following sections will provide a detailed illustration of both attention techniques in passive voice.

Bilinear interpolation is employed to reconstruct the shallow feature in the same dimensional space as the input data. This method can be utilized to achieve this by utilizing it. In comparison to shallow features, deep features have a larger visual field but possess significantly less spatial information.

Channel-Spatial Attention. The channel-spatial attention weight is represented as pink “ \times ” rectangle in Fig. 4.1, which is defined as

$$W_c = \text{softmax}(x^{re} \otimes \varphi^T), \quad (4.1)$$

where $x^{re} \in \mathbb{R}^{3 \times l}$ is the reshaped benign image $x \in \mathbb{R}^{H \times W \times 3}$, as shown in the channel-spatial attention block of Fig. 4.1 with khaki “ \wedge ”, and $\varphi \in \mathbb{R}^{c \times l}$ denotes the reshaped reconstructed feature map, as shown in the channel-spatial attention block with green “ $-$ ”. The reconstructed feature map is obtained by upsampling from the feature map, which is the output of the target DNN in the shallow feature layer with height H' and width W' and channel c , where H' and W' and c depend on the size of the convolution kernel. Here, $\varphi^T \in \mathbb{R}^{l \times c}$ represents the transposed φ , $l = H \times W$, where H and W are the height and width of the benign image, respectively, c denotes the channel depth of the feature map layer, and \otimes represents matrix multiplication. After completing the matrix multiplication of x^{re} and φ^T , the output channel-spatial attention weight $W_c \in \mathbb{R}^{3 \times c}$ is produced by a *softmax* operation, as shown in the channel-spatial attention block with pink “ \times ”.

Pixel-Spatial Attention. The pixel-spatial attention weight is represented as red “ $*$ ” rectangles in Fig. 4.1, which is defined as

$$W_p = \text{softmax}(W_c^{re} \odot x^{re}), \quad (4.2)$$

where $W_c^{re} = W_c \otimes \varphi$ and $W_c^{re} \in \mathbb{R}^{3 \times l}$ represent the reconstructed channel-spatial attention weight, as shown in the pixel-spatial attention block with blue “ $+$ ”. \odot means that matrices W_c^{re} and x^{re} are multiplied by their corresponding elements and then summed up in columns. $W_p \in \mathbb{R}^{1 \times l}$ is the output of the pixel-spatial attention block, marked by red “ $*$ ” in Fig. 4.1.

Owing to the local critical information in the feature map, the pixel-attention weight $W_p \in \mathbb{R}^{1 \times l}$ is reshaped to produce the attention map $W_{map} \in \mathbb{R}^{H \times W \times 1}$, which is adopted to focus the perturbations on the critical classification features of the benign image. The perturbation $\rho \in \mathbb{R}^{H \times W \times 3}$ is then defined as

$$\rho_i = W_{map} \times \frac{\nabla_x J(x^i, y)}{\|\nabla_x J(x^i, y)\|_1}, \quad (4.3)$$

where $i = 1, 2, 3$ corresponds to the RGB channel of a benign image. The adversarial example x^* is defined as

$$x^* = x + \rho. \quad (4.4)$$

Furthermore, an iterative optimization process is implemented to determine the appropriate perturbations, and a momentum-based approach is employed to enhance

Algorithm 4.1: Our method.

Input: A classifier $f(x)$ with loss function $J(x, y)$; the perturbation step length α ; the number of iterations T ; the perturbation threshold ϵ ; benign example x and corresponding label y ; and decay factor μ .

Output: Adversarial example x^* with $\|x^* - x\|_\infty \leq \epsilon$.

```

1  $g_0 = 0; x_0^* = x;$ 
2 while  $1 \leq i < T$  do
3   Input  $x_{i-1}^*$  to  $f$  and obtain the  $\nabla J(x_{i-1}^*, y)$  and feature map;
4   Obtain  $\varphi_{i-1}$  by upsampling feature map;
5   Calculate  $W_c(i-1)$  based on Eq. (4.1);
6   Calculate  $W_p(i-1)$  based on Eq (4.2) and reshape it to  $W_{map}(i-1)$ ;
7   Update  $g_i$  by accumulating the velocity vector in the gradient direction as:
8      $g_i = \mu \cdot g_{i-1} + \frac{\nabla J_x(x_{i-1}^*, y)}{\|\nabla J_x(x_{i-1}^*, y)\|_1} \times W_{map}(i-1)$ 
9   Calculate  $\rho_i = \alpha \cdot g_i$ ;
10  Update  $x_i^* = x_{i-1}^* + \rho_i$ ;
11  if  $\|x_i^* - x\|_\infty > \epsilon$  or  $f(x_i^*) \neq y$  then
12    | break;
13  end
14   $i = i + 1$ ;
15 end
16 return: The adversarial example  $x^* = x_i^*$ .

```

the attack performance. The details of the our method with momentum are presented in Algorithm 4.1, where the loss function $J(x, y)$ is defined as

$$J(x, y) = \begin{cases} \max(-\kappa, Z(x)_y - \max\{Z(x)_{y'} |_{y' \neq y}\}) \\ \quad + \|x - x_0\|_2^2 & \text{untargeted} \\ \max(-\kappa, Z(x)_y - Z(x)_{y'}) \\ \quad + \|x - x_0\|_2^2 & \text{targeted} \end{cases} \quad (4.5)$$

where $\kappa \geq 0$ is a hyperparameter. A large κ means that a high reliability of adversarial examples is required. x_0 denotes the benign image, y'_t denotes the target label, $Z(x)$ is the output confidence value, and $\|x - x_0\|_2^2$ is a regularization term used to constrain the perturbation size.

4.4 Experiments

We primarily study the following contents in our experiment: (1) comparison of untargeted/targeted white-box attacks, (2) comparison of untargeted/targeted black-box attacks, (3) confidence comparison in the attack process, (4) visual analysis of the attack process, and (5) attack analysis of the defensive measures.

4.4.1 Setup

We evaluate our method on the CIFAR10 [24] and ImageNet [25] datasets. For CIFAR10, we use the ResNet32 ($N_{res} = 5$) and ResNet56 ($N_{res} = 9$) models based on the ResNet ($2 + 6 \times N_{res}$) [26], where N_{res} is the number of resblock. For ImageNet, we adopt ResNet-v2 [27], Inception v3 (Inc-v3) [28], and Inception ResNet v2 (IncRes-v2) [29].

Their parameters are set as the default value during the attack. We employed Foolbox [30] to implement various attack methods, including FGSM [9], MI-FGSM [10], PGD [11], BIM [18], DeepFool [19], and C&W [20]. During the attack, we set their parameters to their default values.

In our experiment, we set the perturbation step length $\alpha = 0.001$ and a maximum number of iterations $T = 1000$.

We employed two different defenses. The first is image transformation [31], which falls under the category of input modification defense and is referred to as “Defense 1”. The second defense utilizes a Gaussian blur, which is an add-on network defense technique, namely “Defense 2”, where the parameter σ is 1 and the kernel size is 3*3.

4.4.2 Metrics

In the experiments, the mean value, 0-norm [9], 2-norm [10], and infinite-norm [20] were adopted to evaluate the size of the perturbations, which were expressed as $\bar{\rho}$, $L_0(\rho)$, $L_2(\rho)$, and $L_\infty(\rho)$, respectively. The attack success rate (ASR) is used to measure the attack ability of adversarial examples.

The area ratio of Grad-CAM (ARGC) is designed to quantify the object area reduction of different attack methods:

$$\text{ARGC}(x) = \frac{\text{SNg}(x^*) \cap \text{SNg}(x)}{\text{SNg}(x)}, \quad (4.6)$$

where $\text{SNg}(x)$ is a binary mask indicating the image pixels with a weight above the threshold τ . $\text{SNg}(x) = \text{sign}(\text{Norm}(g(x)), \tau)$, where $\text{Norm}(g(x))$ function normalizes $g(x)$ between 0 and 1, and thus $\text{ARGC}(x) \in [0, 1]$. $\text{sign}(x, \tau)$ denotes the sign function with the threshold τ , as follows:

$$\text{sign}(x, \tau) = \begin{cases} 1, & x > \tau \\ 0, & x \leq \tau \end{cases} \quad (4.7)$$

4.4.3 Performance of Our Method for White-Box Attack

The white-box evaluation results of our method are shown in Fig. 4.3. The attack performance is evaluated by the mean value and the L_p norm of the perturbations, ASR, and mean time of generating one adversarial example. Based on the evaluation results in Fig. 4.3, the following conclusions can be drawn. Our method demonstrates comparable performance to C&W in terms of attack success rate (ASR) and the magnitude of perturbations, while also outperforming C&W in terms of attack speed. Additionally, our method exhibits a relatively stable attack performance on ImageNet against various target models.

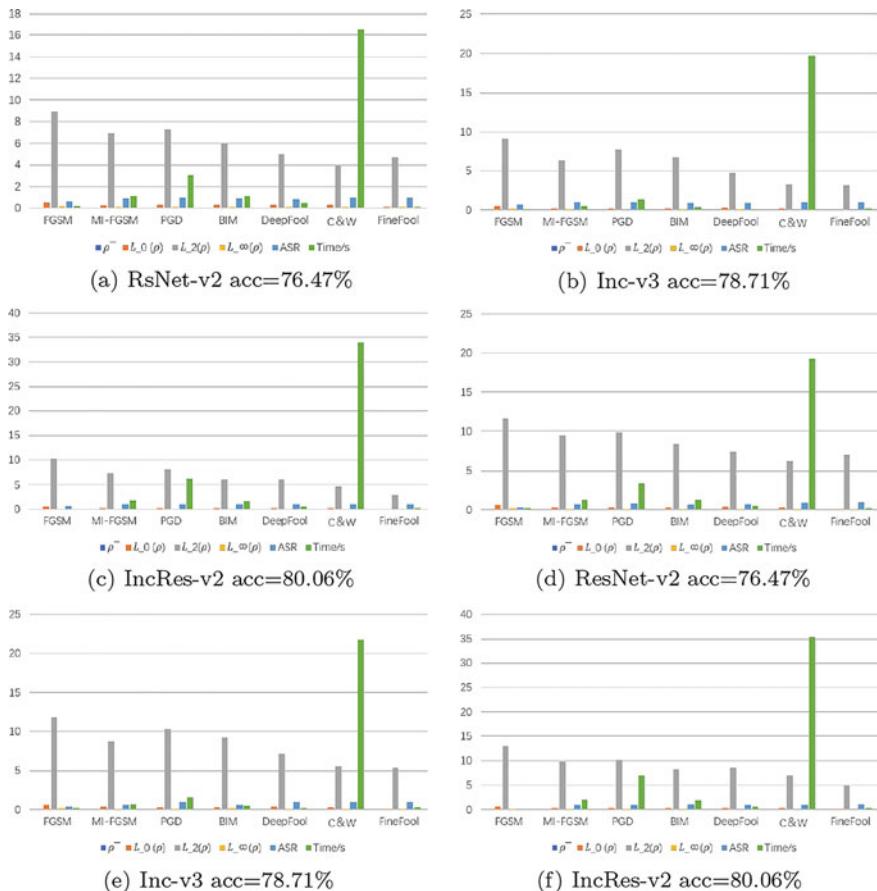


Fig. 4.3 The comparison of white-box attack results against different DNN models on ImageNet. Subfigure **a-c** are the adversarial examples obtained by untargeted attacks. Subfigure **d-f** are the adversarial examples obtained by targeted attacks

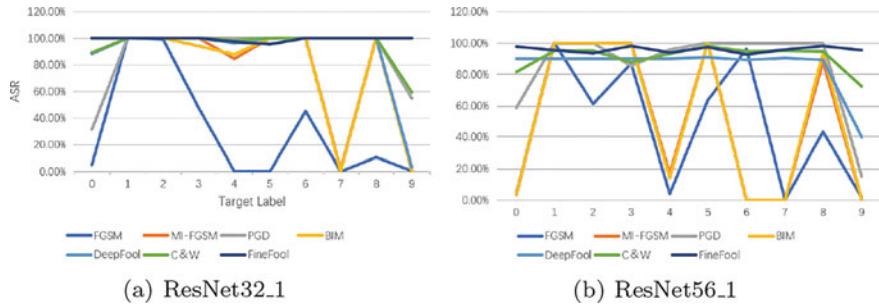


Fig. 4.4 The comparison of targeted attack results against different DNN models on CIFAR10 (white-box attack)

By focusing on deep feature information, our method is better able to attack complex datasets and complex DNNs. For targeted attacks, our method can clearly differentiate between perturb locations that need to be added in order to attack different target tags.

4.4.3.2 Further Analysis of the White-Box Attack

Figure 4.4 provides the white-box targeted attack performance of different attack methods. For targeted attacks, the attack difficulty depends on the target labels. Figure 4.4 shows the ASR of each target label of different targeted attacks, and the number of target labels with ASR greater than 85% and greater than 95%.

According to the results in Fig. 4.4, the following conclusions can be drawn. The targeted attack difficulty varies for different pre-set target labels. Different target DNN models exhibit consistency against different target labels, meaning that there exist robust and vulnerable target classes in a dataset that are strongly related to the attack methods while weakly related to the target DNN models. When generating adversarial perturbations for different target labels, our method can focus on the perturbations of the corresponding target label, ensuring the stability of the ASR in different target labels.

4.4.4 Performance of Our Method for Black-Box Attack

4.4.4.1 Analysis of Black-Box Untargeted Attack

It is known that the black-box attack is more difficult than the white-box attack. Substitute models are usually adopted as the target model to implement black-box attacks. Figure 4.5 provides the comparison between the black-box untargeted attack results, which evaluates the attack performance via the mean value of perturbations and ASR, where the mean perturbations are calculated by the white-box attack against the substitute model.

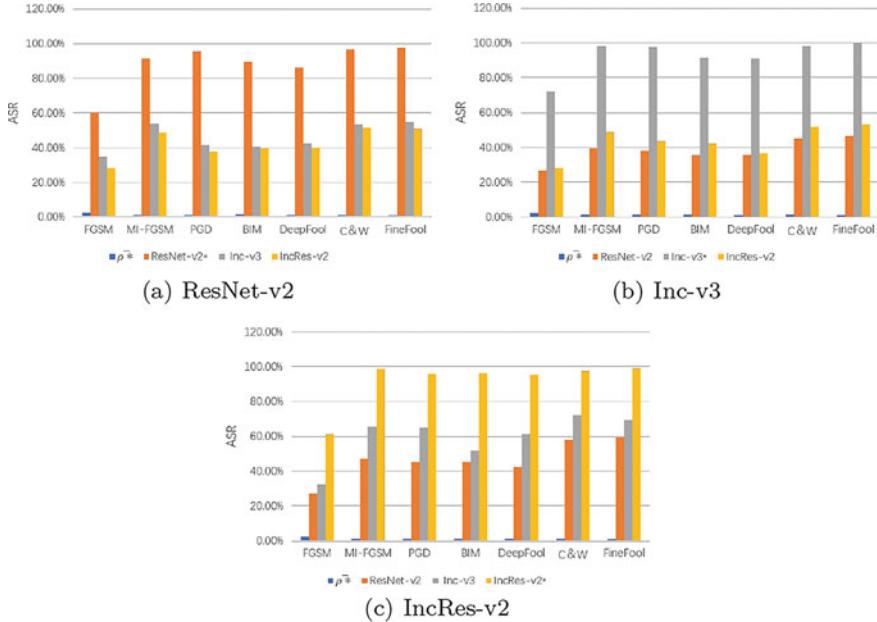


Fig. 4.5 The comparison of white-box attack results against different DNN models on MNIST, CIFAR10, and ImageNet

According to the results in Fig. 4.5, we can conclude that the transferability of our method is better than that of the baselines in black-box untargeted attacks. In the black-box attacks, the substitute models we designed are similar to the target DNN models. Therefore, our method could use the information of vulnerable neurons learned from substitute models to produce effective adversarial examples for implementing black-box attacks against the target DNN models. Our method could also take advantage of the channel-spatial attention to collect more vulnerability information, thus realizing a better transferability compared to that of the baselines.

4.4.4.2 Analysis of Black-Box Targeted Attack

Figure 4.6 provides the comparison of black-box targeted attack results on CIFAR10 against different DNN models, which evaluates the attack performance via the mean value of perturbations and ASR. According to the results in Fig. 4.6, we conclude that the transferability of our method is better than that of the baselines in black-box targeted attacks. The ASR values of our method against different target DNN models were all better than those of the baselines in black-box targeted attacks on the CIFAR10 dataset. The reasons why our method could reach these effective attack

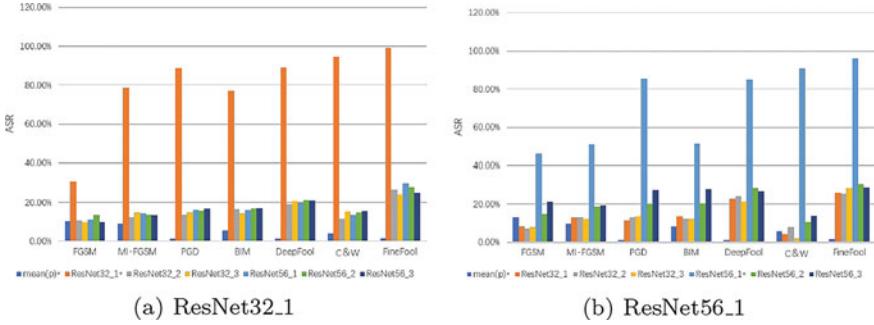


Fig. 4.6 The comparison of black-box targeted attack performance based on substitute models

results are as follows. Our method would implement the white-box targeted attack while paying close attention to the vulnerability of a target DNN model as much as possible. Therefore, better transferability can be reached when performing a black-box targeted attack based on substitute models.

4.4.5 Visualization Analysis

4.4.5.1 Analysis of Different Attacks Against the Same DNN

Figure 4.2 shows the adversarial examples and the Grad-CAM maps of the ResNet-v2 model attacked by different methods. Different attack methods have narrowed the scope of the DNN’s attention to object areas. To illustrate the situation in Fig. 4.2 more accurately, we randomly selected 200 benign examples from ImageNet and attacked ResNet-v2 with different methods to generate their adversarial examples. The ARGC metric corresponding to adversarial examples and benign ones were calculated; they are as shown in Fig. 4.7. It can be seen that the ARGC values of our method are the smallest under different τ parameters, which could maximize the impact of the model’s attention to the object areas.

4.4.5.2 Analysis of Our Method Against Different DNNs

As shown in Fig. 4.3, we can see that ResNet-v2 and IncRes-v2 were more vulnerable to our method than Inc-v3. Figure 4.8 provides a visual explanation. Compared to ResNet-v2 and IncRes-v2, Inc-v3 continued to maintain a larger scope of attention to the object areas during the attack process, which made it difficult for them to be attacked.

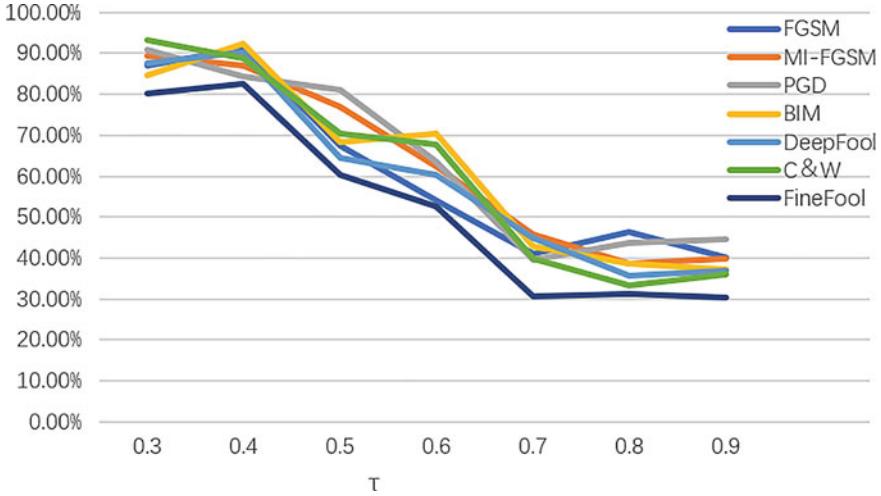


Fig. 4.7 The ARGC metric of different attacks against ResNet-v2 model under different τ

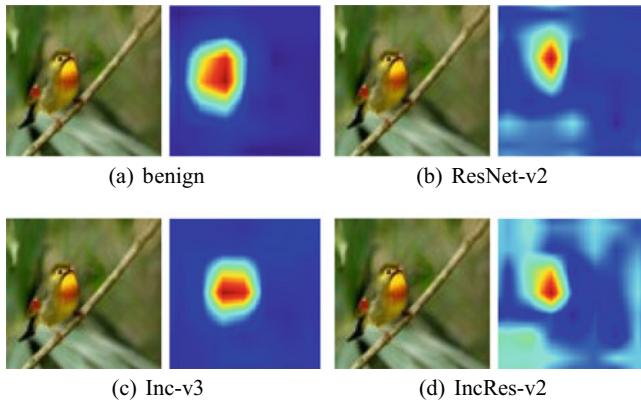


Fig. 4.8 The attention region visualization of adversarial examples produced by our method against different DNNs. In subfigure **a**, the left is benign example and the right is Grad-CAM map of benign example. From subfigures **b**, **c**, and **d** are the adversarial examples obtained by our method against different DNNs and their Grad-CAM maps

4.4.6 Confidence Analysis

The normalized logits of the target DNN model output can measure the attack reliability of adversarial examples. The logits value is defined as the output of the target model before activation. In general, for an adversarial example, the lower the confidence value of the ground-truth label, the higher the confidence value of the adversarial label of the untargeted attack (or the target label of the targeted attack), and the higher the attack reliability. Figures 4.9 and 4.11 show the visualization of

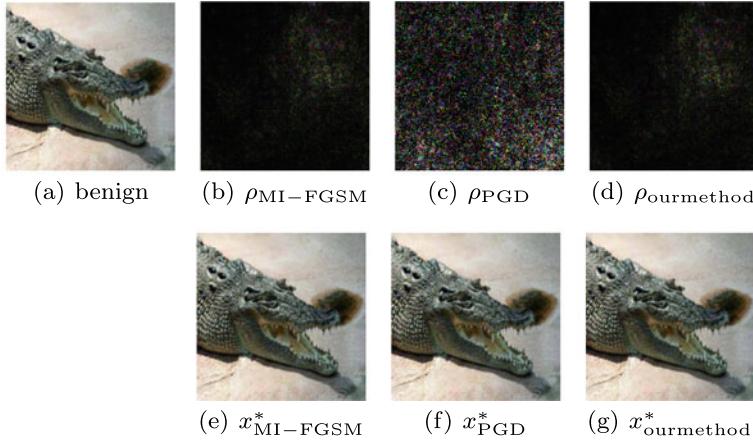


Fig. 4.9 The example of different attack methods against Resnet-v2. And all perturbations are magnified 50 times for better visualization. The subfigure **a** is the benign example. The subfigures **b**, **c**, and **d** are the adversarial perturbation produced by MI-FGSM, PGD, and our method, respectively. The subfigures **e**, **f**, and **g** are the adversarial examples crafted by MI-FGSM, PGD, and our method against Resnet-v2 model, respectively

perturbations and adversarial images from ImageNet, which are obtained by different attack methods against different target DNN models, such as Resnet-v2 and Inc-v3. Figures 4.10 and 4.12 show the logit curves of different attacks in the iterative process corresponding to Figs. 4.9 and 4.11, respectively. The solid line with “.” denotes the curve change of the logit corresponding to the label of adversarial examples (such as “our method(adv)”), and the dashed line represents the curve change of the logit corresponding to the ground-truth label of benign examples (such as “our method(ben)”). Base on the logit outputs of adversarial examples produced by different attacks, we find that the mean confidence levels of adversarial examples generated by PGD and MI-FGSM were the highest out of the top 2 in all baselines, though the ASR values of PGD and MI-FGSM were not the top 2. Therefore, PGD and MI-FGSM were selected in Figs. 4.9, 4.10, 4.11, and 4.12.

According to the results in Figs. 4.9, 4.10, 4.11, and 4.12, the costs of perturbations produced by different attack methods were different. The attack ability of adversarial examples could be improved by the perturbations added in each iteration step. The decrease in the output logits of the ground-truth label and the increase in the output logits of the adversarial label reflected the performance of perturbations in each iteration step. The perturbation produced by PGD is solely based on the gradient information, leading to significant fluctuations in the logit curve in each iteration step. MI-FGSM utilizes the momentum information to adjust the search direction of perturbations, resulting in smaller fluctuations in the logit curve compared to PGD. The ground-truth label and adversarial label had the largest logit distance after the attack, as the logit curve of the adversarial label produced by our method showed an upward trend.

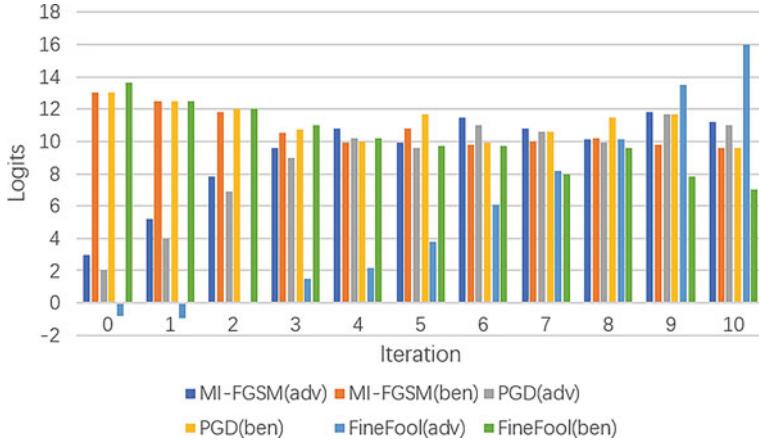


Fig. 4.10 The logits curve of different attack methods against Resnet-v2 in iterations. The abscissa is iteration number, and the ordinate is the logits value. The logits value represents the output of the target model before activation. If the logits value of the adversarial example is high, and the logits difference between the adversarial example and the benign one is large, then the attack method is reliable and effective

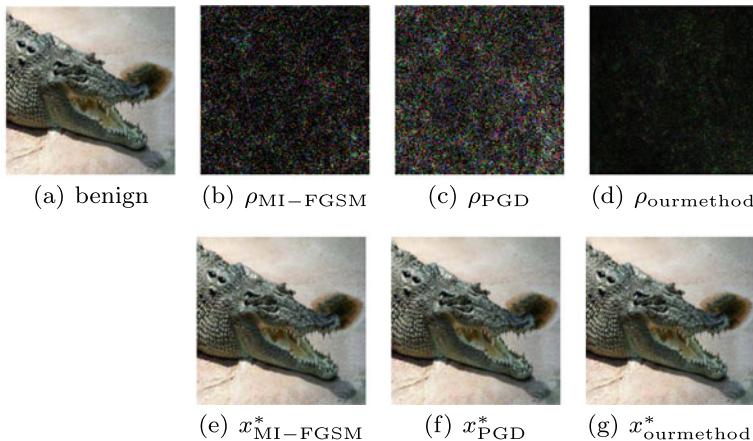


Fig. 4.11 The example of different attack methods against Inc-v3. And all perturbations are magnified 50 times for better visualization. The subfigure **a** is the benign example. The subfigures **b**, **c**, and **d** are the adversarial perturbation produced by MI-FGSM, PGD, and our method, respectively. The subfigures **e**, **f**, and **g** are the adversarial examples crafted by MI-FGSM, PGD, and our method against Inc-v3 model, respectively

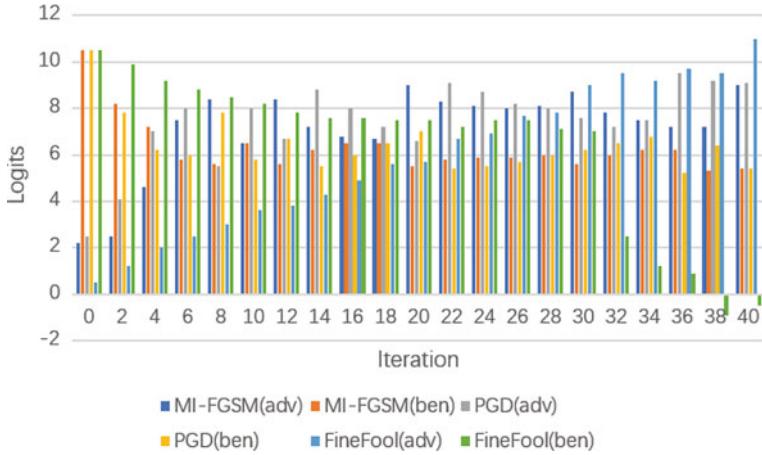


Fig. 4.12 The logits curve of different attack methods against Inc-v3 in iterations. The abscissa is iteration number, and the ordinate is the logits value. The logits value represents the output of the target model before activation. If the logits value of the adversarial example is high, and the logits difference between the adversarial example and the benign one is large, then the attack method is reliable and effective

The final logit value of the adversarial example produced by our method was the highest against different target DNN models, while the final logit value of the benign image was the lowest. The logit curve showed a linear trend and converged. Furthermore, the convergence speed and the number of iteration steps for different attack methods were influenced by the structural complexity of target DNN models.

The above analysis suggests that our method was found to be more reliable and effective, resulting in smaller overall attack costs compared to the baselines. For each attack, we calculated the mean logits value of 1,000 adversarial examples and corresponding benign ones on the ImageNet dataset. For the Resnet-v2 model, the mean logits values of the adversarial examples produced by MI-FGSM, PGD, and our method were 11.98, 12.50, and 18.35, respectively. The corresponding mean logits differences are 1.83, 1.76, and 8.77, respectively. For the Inc-v3 model, the mean logits of adversarial examples crafted by MI-FGSM, PGD, and our method are 8.11, 7.42, and 10.95, respectively. The corresponding mean logits differences are 3.87, 2.05, and 9.25, respectively. We can conclude that the mean logits and mean logits differences of adversarial examples crafted by our method are the highest compared with baselines, which shows the reliability and effectiveness of our method. The reasons why our method could achieve these performance levels are as follows: In comparison to the momentum-based direction modifications of perturbations, our method was able to concentrate on the effectiveness of perturbations in each step through channel-spatial attention and pixel-spatial attention. By updating the input through backpropagation, it resulted in an increasing trend of the logit curve.

4.4.7 Attack Comparison Against Defense

The significance of adversarial attacks lies not only in identifying the vulnerabilities of target DNN models but also in measuring the defensive capabilities of defense methods. Figure 4.13 presents an ASR comparison of the white-box untargeted attack results produced by various attack methods against the Inc-v3 model with different defenses.

The results in Fig. 4.13 indicate that our method exhibits a superior attack effect against input modification and network add-on defenses when compared to the corresponding baselines. This is due to the following reasons. By focusing on the vulnerability of target DNN models, our method is able to generate adversarial perturbations that are close to the object contour, which is crucial for deep model recognition. Therefore, our method is capable of generating adversarial perturbations that are difficult to eliminate for the target DNN model with input modification defense. Furthermore, our method is able to investigate input-activated neurons and add perturbations to achieve incorrect activations for the adversarial attack. Therefore, the effect of the network add-on defense against our method's searching adversarial perturbations was also smaller compared to the baselines.

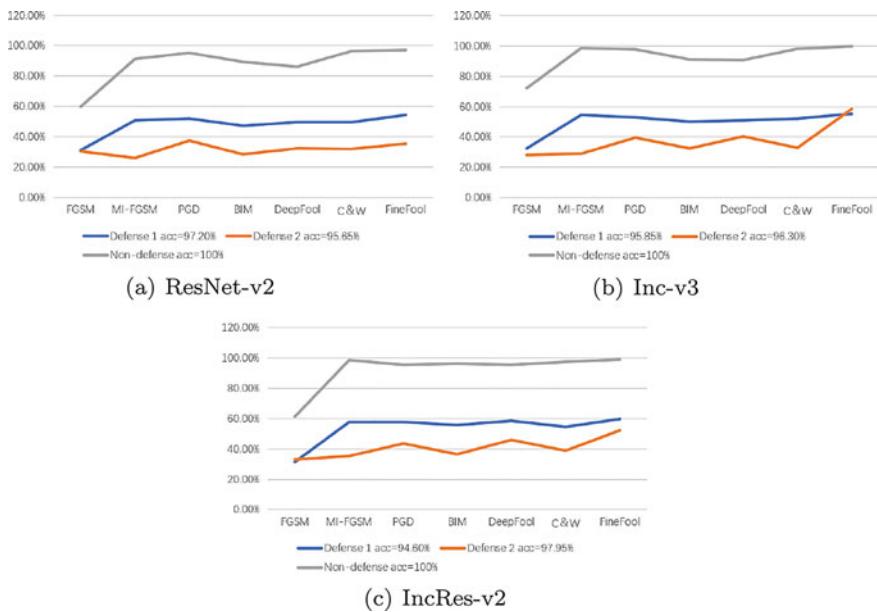


Fig. 4.13 The white-box untargeted attack results of different attack methods against different deep models (acc = 100% for selected benign images) with different defenses on ImageNet dataset

4.5 Conclusion

In this chapter, a novel attention-based method for generating adversarial examples. During the research on adversarial attack, it was observed through visualization that there are strong correlations between the perturbations and the object contour. To produce more effective adversarial examples with smaller perturbations, the features were focused onto the object contour. The key to our method was the channel-spatial attention and the pixel-spatial attention, which focused on the channel-feature distribution and pixel-region distribution, respectively. The former reduced the area of investigation by the DNN while the latter achieved the error location of the object contour. From the analytical explanation and experimental analysis, we conclude that our method is capable of generating highly effective adversarial examples with better attack performance but fewer perturbations.

References

1. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–14 (2015)
2. Chen, J., Zheng, H., Xiong, H., Wu, Y., Lin, X., Ying, S., Xuan, Q.: Dgepn-gcen2v: a new framework for mining GGI and its application in biomarker detection. *Sci. China Inf. Sci.* **62**(9), 1–3 (2019)
3. Young, T., Hazarika, D., Poria, S., Cambria, E.: Recent trends in deep learning based natural language processing. *IEEE Comput. Intell. Mag.* **13**(3), 55–75 (2018)
4. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey. *IEEE Access* **6**, 14410–14430 (2018)
5. Dennis, A.R., Minas, R.K.: Security on autopilot: Why current security theories hijack our thinking and lead us astray. *Data Base Adv. Inf. Syst.* **49**(s1), 15–37 (2018)
6. Oh, S.L., Hagiwara, Y., Raghavendra, U., Yuvaraj, R., Arunkumar, N., Murugappan, M., Acharya, U.R.: A deep learning approach for parkinson’s disease diagnosis from eeg signals. *Neural Comput. Appl.* **32**(15), 10927–10933 (2020)
7. Kwon, H., Kim, Y., Park, K.W., Yoon, H., Choi, D.: Friend-safe evasion attack: An adversarial example that is correctly recognized by a friendly classifier. *Comput. Secur.* **78**, 380–397 (2018)
8. Cisse, M., Adi, Y., Neverova, N., Keshet, J.: Houdini: Fooling deep structured visual and speech recognition models with adversarial examples. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4–9 December 2017, Long Beach, CA, USA, pp. 6978–6988 (2017)
9. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–10 (2015)
10. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp. 9185–9193. IEEE Computer Society (2018)
11. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–28. OpenReview.net (2018)

12. Chen, P., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.: Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017, pp. 15–26. ACM (2017)
13. Sarkar, S., Bansal, A., Mahbub, U., Chellappa, R.: UPSET and ANGRI: breaking high performance image classifiers, pp. 1–8 (4 July 2017)
14. Chen, J., Zheng, H., Xiong, H., Shen, S., Su, M.: Mag-gan: massive attack generator via gan. *Inf. Sci.* **536**, 67–90 (2020)
15. Chen, J., Su, M., Shen, S., Xiong, H., Zheng, H.: Poba-ga: perturbation optimized black-box adversarial attacks via genetic algorithm. *Comput. Secur.* **85**, 89–106 (2019)
16. Miyato, T., Dai, A.M., Goodfellow, I.: Adversarial training methods for semi-supervised text classification. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, pp. 1–11. OpenReview.net (2017)
17. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial machine learning at scale. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, pp. 1–17. OpenReview.net (2017)
18. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, pp. 1–14. OpenReview.net (2017)
19. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: A simple and accurate method to fool deep neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 2574–2582. IEEE Computer Society (2016)
20. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017, pp. 39–57. IEEE Computer Society (2017)
21. Chen, J., Zheng, H., Chen, R., Xiong, H.: Rca-soc: a novel adversarial defense by refocusing on critical areas and strengthening object contours. *Comput. Secur.* **96**, 101916.1–101916.18 (2020)
22. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: visual explanations from deep networks via gradient-based localization. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017, pp. 618–626. IEEE Computer Society (2017)
23. Zhang, H., Goodfellow, I., Metaxas, D., Odena, A.: Self-attention generative adversarial networks. In: Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA, vol. 97, pp. 7354–7363. PMLR (2019)
24. Alex, K.: Learning Multiple Layers of Features from Tiny Images, pp. 1–60. Computer Science Department, University of Toronto, Technical Report (2009)
25. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: a large-scale hierarchical image database. In: 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20–25 June 2009, Miami, Florida, USA, vol. 1-4, pp. 248–255. IEEE Computer Society (2009)
26. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 770–778. IEEE Computer Society (2016)
27. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Computer Vision–ECCV 2016–14th European Conference. Amsterdam, The Netherlands, October 11–14, 2016, vol. 9908, pp. 630–645. Springer, Berlin (2016)
28. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 2818–2826. IEEE Computer Society (2016)

29. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA, pp. 4278–4284. AAAI Press (2017)
30. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Foolbox tool (2018). <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks>
31. Guo, C., Rana, M., Cisse, M., van der Maaten, L.: Countering adversarial images using input transformations. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–12. OpenReview.net (2018)

Chapter 5

Query-Efficient Adversarial Attack Against Vertical Federated Graph Learning



5.1 Introduction

Federated Graph Learning (FGL) is an innovative distributed learning method that constructs Graph Neural Networks (GNNs) using distributed data while keeping it decentralized to preserve privacy. FGL has two categories: Horizontal FGL (HFGL) and Vertical FGL (VFGL), which are determined by the data distribution characteristics. HFGL [1–3] is designed for clients who share the same feature space but have different node ID spaces. In a vertical configuration, VFGL [4, 5] is suitable when clients share the same node ID space, but have different feature spaces.

In this chapter, we explore the vulnerability of VFGL by focusing on adversarial attacks. Numerous attacks have been proposed against centralized GNNs (i.e., the GNN with centralized data), including white-box attacks [6–9], gray-box attacks [10, 11], and black-box attacks [12–15]. While successful attacks have been conducted on centralized GNNs, they face challenges in VFGL scenarios. White-box/gray-box attacks are not applicable to VFGL as they require knowledge of the target model's structure, parameters, or both. In VFGL, the malicious client does not have access to the server model, which makes the final decision. When conducting black-box attacks, a common approach is to create a shadow model, which is then used to generate adversarial examples that are transferred to the target model. Another type of black-box attack involves optimizing adversarial examples based on feedback from the target model. However, this method requires many queries to the server model, leading to high query costs and a greater risk of being detected. Therefore, we have chosen to use the shadow model strategy for our adversarial attacks against VFGL.

In summary, there are some challenges for adversarial attacks against VFGL. (i) *Knowledge Limitation*: It requires constructing successful attacks without any structure or parameter details of the server model. (ii) *Query Limitation*: The server should limit prediction query to improve stealthiness. Our proposed framework is an adversarial attack on VFGL via a four-stage pipeline to address these challenges, as shown in Fig. 5.1. In order to address challenge (i), we create a trustworthy shadow

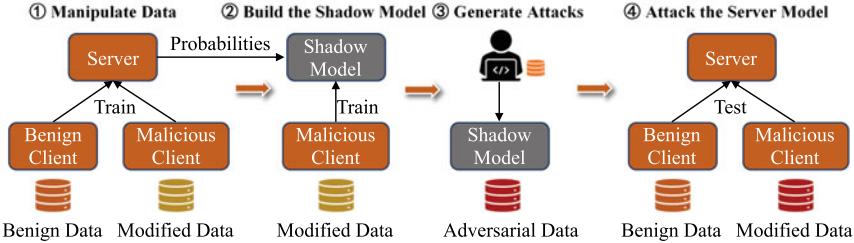


Fig. 5.1 The pipeline of query-efficient hybrid adversarial attack framework against VFGL

model while training. This is because the training data is private to each client and the server, which means a malicious client could modify their data to make the server model rely more on their information. This would be advantageous to the malicious client as it would allow them to train a shadow model similar to the server model, which they could then use to generate adversarial examples without having to ask the server for permission. This helps to tackle challenge (ii).

In addition, manipulation of the training data should address the following challenges. (iii) *Attack Target Limitation*: The target of the attacker may depend on the manipulated data, i.e., the attacker is unable to attack arbitrary desired nodes in flexible manners. (iv) *Benign Performance Degradation*: Data manipulation may cause benign performance degradation in the inference stage. To tackle challenge (iii), data manipulation is used to increase the contribution of the malicious clients, but not to leave a specific backdoor in the GNN, i.e., it enables the attacker to generate any desired adversarial examples for different target nodes. To address the challenge (iv), the manipulated training VFGL should keep good task performances for benign examples. It has been observed that the model predictions are influenced by the neural paths made up of key neurons, and distinctive portions of the model are activated by different classes [16]. Consequently, we propose to use the key neural path of the malicious client's local model to find the significant node features corresponding to the task for manipulation, which preserves the satisfying performance for benign examples.

The contributions of this work are summarized as follows:

- As far as we know, this is the initial attempt at an adversarial attack on VFGL. To combat the inefficiency of centralized attacks on VFGL, we have introduced a new framework for query-efficient adversarial attacks called our method. This framework has shown a considerable improvement in centralized adversarial attacks.
- Inspired by the observation that the server model in VFGL is more likely to be misled by high-contribution clients, a training data manipulation strategy is adopted for our method to enhance the adversarial attack. Thus, a novel hybrid attack is proposed. It involves manipulating training data and generating adversarial attacks during testing.
- By constructing a reliable shadow model within the malicious client, our method becomes more efficient and effective, requiring only one query to the server.

- The study tested our method on four real-world graph-structured datasets and evaluated its performance against four advanced adversarial attacks in VFGL. The results showed that our method has achieved state-of-the-art performance and still poses a threat to VFGL, even with its potential defense mechanism. The effectiveness of our method was also demonstrated through the identification of sensitive neurons and visualization of t-SNE.

This chapter is structured as follows. Section 5.2 provides a review of related works on federated graph learning, adversarial attacks on GNNs, and privacy leakage on graph learning. Section 5.3 presents the preliminary and problem formulation. The details of our method are then shown in Sect. 5.4, and the evaluation of our method is presented in Sect. 5.5. At last, the chapter is concluded in Sect. 5.6.

5.2 Related Work

Studies have shown that GNNs are vulnerable to imperceptible adversarial attacks, which can be categorized based on the attacker's knowledge and capabilities as white-box or gray-box attacks. Studies have shown that GNNs are vulnerable to imperceptible adversarial attacks, which can be categorized based on the attacker's knowledge and capabilities as white-box or gray-box attacks.

White-Box Attacks: In the white-box setting, the attacker has the parameters of the target model, training data, and real data. The gradient-based attack method is a typical white-box attack [6–9, 17], which applies the gradient information of the target model to generate the adversarial perturbations.

Gray-Box Attacks: In the gray-box setting, the attacker attempts to train the shadow model to approximate the target model. For instance, Zügner et al. [10] trained a simplified graph convolutional network for Nettack, which computes the misclassification loss sequentially for each edge in the candidate set. Besides, Metattack [11] is proposed based on the meta-gradient of the shadow model.

Black-Box Attacks: Reinforcement learning is one of the most widely used techniques in black-box attacks. Dai et al. [12] proposed RL-S2V, which applies the reinforcement learning and only requires the prediction labels from the target model. Also, genetic algorithm is applied in the black-box settings where the prediction is available [12]. Ma et al. [15] proposed a reinforcement learning-based attack method named RaWatt, which preserves the properties of the graph. Chang et al. [13] proposed GF-Attack, which performs the attack on the graph filter without accessing any knowledge of the target model. Ma et al. [14] extended the common gradient-based attack to black-box settings via the relationship between gradient and PageRank.

Most of the adversarial attacks are based on the modification of the edges and characteristics of the nodes. However, graph injection attacks focus on injecting malicious nodes into the graph rather than modifying existing edges and features [18–21].

5.3 Preliminary and Problem Formulation

First, we define VFGL and attack it. Then the threat model is analyzed.

5.3.1 Vertical Federated Graph Learning

Let $G = (V, E)$ be a graph with nodes set V and edges set E . Define A as the graph's adjacency, and X as its properties. In VFGL, each client uses its own private data to maintain the local GNN, and trains the server model together. It should be noted that in the existing works [4, 5], the edges and node features are split for each client. However, the split subgraph is not in line with the real distribution [22]. Therefore, we propose a method that only segment the features of nodes, but not segment them, in order to maintain its original distribution characteristics. The server assembles the local nodes uploaded by each client into an embedded set. In this chapter, we consider concatenating the local embeddings as the aggregation strategy:

$$\begin{aligned} h_{global} &\leftarrow \text{CONCAT}(h_1, \dots, h_i, \dots, h_K) \\ \text{s.t. } h_i &= f_\theta^i(A_i, X_i, V), \end{aligned} \tag{5.1}$$

where K is the number of clients in VFGL, h_i is the i -th client's local embeddings produced by the local GNN $f_\theta^i(\cdot)$ with the local private data (A_i, X_i) .

Then, h_{global} is utilized to train the server model $\mathcal{S}(\cdot)$:

$$\begin{aligned} \mathcal{L}_{train} &= - \sum_{n=1}^{|V_L|} \sum_{l=1}^{|F|} Y_{nl} \ln(Y'_{nl}) \\ \text{s.t. } Y' &= \mathcal{S}(h_{global}), \end{aligned} \tag{5.2}$$

where V_L denotes the subset of V labeled with the ground truth, $|F|$ is the number of node classes. Y is the label list and Y' is the probabilities list that is predicted by the server model. Here, a L -layers multilayer perceptron (MLP) is applied as the server model in VFGL in this chapter.

5.3.2 Adversarial Attack on Vertical Federated Graph Learning

In the centralized graph-learning environment, attackers often manipulate the input data to reduce the embedding quality of nodes, thus attacking the target GNN. Similarly, malicious clients can manipulate local data, upload damaged embedded programs, and carry out malicious attacks on server mode:

$$\begin{aligned}
\hat{Y}'_t &= \mathcal{S}(\hat{h}_{global}) \\
s.t. \quad \hat{h}_{global}^{v_t} &\leftarrow CONCAT(h_1^{v_t}, \dots, \hat{h}_m^{v_t}, \dots, h_K^{v_t}) \\
\hat{h}_m^{v_t} &= f_\theta^m(\hat{A}_m, \hat{X}_m, v_t),
\end{aligned} \tag{5.3}$$

where f_θ^m is the local GNN of malicious client, \hat{A}_m and \hat{X}_m are perturbed data. $\hat{h}_m^{v_t}$ is denoted as the low-quality node embeddings of the target node v_t , which is uploaded by the malicious client.

The target of the malicious client is maximizing the loss of v_t :

$$\begin{aligned}
\max_{v_t \in T} \sum \mathcal{L}_{atk}(\mathcal{S}(\hat{h}_{global}^{v_t}), y_{v_t}) \\
s.t. \quad \mathcal{L}_{atk} = - \sum_{l=1}^{|F|} Y_{v_t l} \ln(Y'_{v_t l}),
\end{aligned} \tag{5.4}$$

where y_{v_t} is the ground truth of v_t , and T is the target node set.

5.3.3 Threat Model

Scenario: In this chapter, multiple clients collaborate to train a central server model for node classification in VFGL. During the training process, the server model sends personalized gradient information and predictions to each client. In the testing process, the server model only provides the querying service and returns the probability. In both the training and testing processes, the server model's structure and inner parameters are kept hidden from clients.

Adversary knowledge: It is assumed that a malicious client only has access to its own data, such as the adjacency, node features, and ground-truth label of training nodes. It is practical for the client to not collect extra information from other clients in the real world. The client is only allowed to manipulate its own data in order to attack the server model.

Attack goal: The aim of a malicious client is to deceive the server model by uploading adversarial node embeddings in order to provide incorrect predictions to the benign clients. It is important to note that we only focus on creating adversarial node embeddings using local GNN and not by constructing fake node embeddings directly. This is because generating fake node embeddings requires high-quality feedback from the server model, which requires multiple queries.

5.4 Methodology

As mentioned in Sect. 5.3.2, the malicious client can manipulate edges, features, or both. In this chapter, our method focuses on modifying the features of the training nodes, due to modifying features only affecting a few features rather than all dimensions (modifying edges) during the aggregation, which is more covert. In addition, modifying partial but not all the node features can preserve the performance of the main task as much as possible. The proposed method follows three stages: (i) manipulate the local data according to neuron testing results (Sect. 5.4.1); (ii) build the shadow model by using the manipulated data (Sect. 5.4.2); and (iii) generate the adversarial attacks via the shadow model (Sect. 5.4.3). The overview of our method is shown in Fig. 5.2 and the pseudocode is given in Algorithm 5.1. We detail each step in the following subsections.

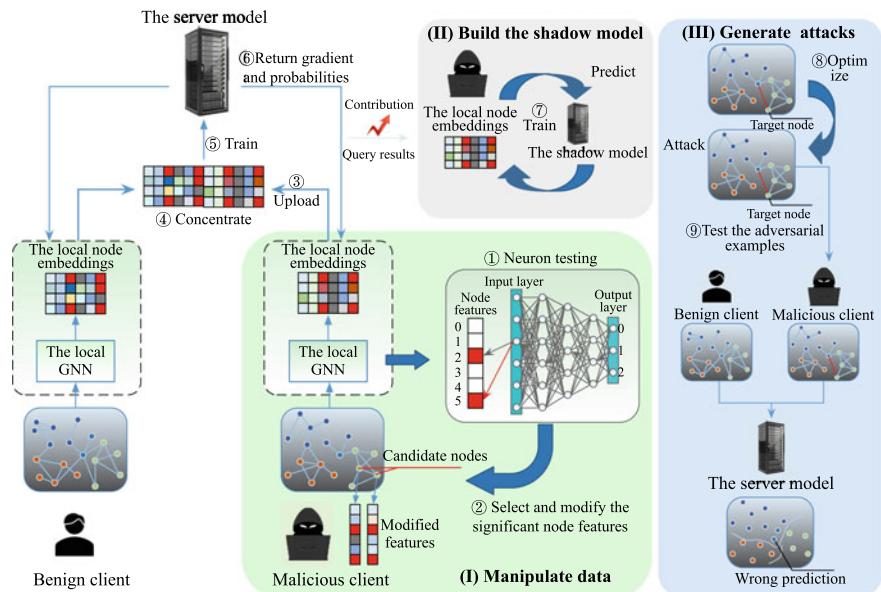


Fig. 5.2 Framework of our method. It can be divided into three stages: (i) Manipulate the local data according to neuron testing results. (ii) Build the shadow model by using the manipulated data. (iii) Generate the adversarial attacks via the shadow model

5.4.1 Data Manipulation

In order to get the candidate nodes and characteristics, the algorithm is divided into two steps: (i) The important neural path is located and the candidate node is acquired; (ii) the target characteristics of the candidate node are selected and modified.

5.4.1.1 Locate Significant Neural Path

In order to suppress the degradation of main task performance as much as possible, our method focuses on locating important neural pathways related to the main task and processing the corresponding data. First, using a L -layers fully connected DNN as an example, we define the neural pathway as follows:

Definition 5.1 The neural path starts from the input neuron, traverses the hidden layer, and ends with the neuron in the output layer. The neural path P can be expressed as

$$P = [a^1, b^2, \dots, n^L], \quad (5.5)$$

where a, b, \dots, n are the indexes of any neuron in the 1-st, 2-nd, ..., L -th layer of the DNN, respectively.

Then, we introduce the neurons in GNN for a better understanding of our method. Take graph convolutional network (GCN) [23] as an example, the output of the l -th hidden layer (without activation function) can be described as

$$Z^l = H^{l+1} = \tilde{D}^{-1/2}(A + I_N)\tilde{D}^{-1/2}H^lW^l, \quad (5.6)$$

where \tilde{D} is the degree matrix of $A + I_N$, I_N is an identity matrix and $H^0 = X$. Denote $W \in \mathbb{R}^{N \times M}$, where N is the number of nodes in the graph and M is the number of neurons in the l -th hidden layer. Thus, the output of the i -th neuron for a node t is the i -th column of Z_t^l , which is denoted as $[Z_t^l]_i$.

To locate the most important neurons for node classification, this process is performed after training for a number of epochs τ , at which time the corresponding relationship between neurons and downstream tasks is established. More details about the number of epochs τ will be discussed later. Then, for a L -layers GNN, the first neuron of the node t is started with the output layer:

$$k^l = \arg \max_{k^l} [|Z_t^l|]_{k^l}, \quad (5.7)$$

where $|\cdot|$ is the absolute value operator.

The rest significant neurons are selected by the following rule:

$$k^{l-1} = \arg \max_{k^{l-1}} \left[\left| \frac{\partial [Z_t^l]_{k^l}}{\partial Z_t^{l-1}} \cdot Z_t^{l-1} \right| \right]_{k^{l-1}}. \quad (5.8)$$

Our method traverses all layers of the local GNN according to the above rule, and finally obtains the node t 's reverse neural path $P_t = [k^l, k^{l-1}, \dots, k^1]$. Next, the train set is divided according to the node categories. For each category of nodes, we perform neuron tests on them and count the number of nodes with the same neural path. Then, take the neural path with the most nodes as the target path P_T , and add the rest nodes corresponding to other paths into the candidate nodes set C .

5.4.1.2 Select and Modify the Target Feature

After obtaining the valid neuron path P_T , we can identify the valid neuron at the local GNN input layer k^1 through P_T . Then extract the k^1 -th column of the weight matrix W^1 from the first layer of the local GNN, which is represented as w_k^1 . The top M index of w_k^1 is selected from the budget $M = \gamma \cdot d$ (number of features to be modified), which is considered the target feature set I . γ and d are the size of the feature changes and the number of node features, respectively. Finally, change the i -th dimension of the candidate node's attributes to

$$\begin{aligned} \tilde{X}_c^i &= \max(X_c) \\ \text{s.t. } c \in C \text{ and } i \in I. \end{aligned} \tag{5.9}$$

Since we believe that the target feature is used in the path of the neural network for maximum impact, we replace the target feature with the maximum of the original feature X_c .

5.4.2 Shadow Model Construction

Instead of frequent queries, constructing a shadow model can provide attackers with useful information, such as gradient information that is widely adopted in adversarial example generation but unavailable in VFGL. First, we construct the shadow model $\tilde{\mathcal{S}}$ by cascading the local GNN and a L -layers MLP. Then we query for probabilities p which are used to train the shadow model. The probabilities contain not only the essential information of the main task, but also the behavior of the server model. Thus, we consider the mean square error (MSE) as the target of the shadow model during the training process:

$$\mathcal{L}_{\tilde{\mathcal{S}}} = \frac{1}{R \times |F|} \sum_{i=1}^R \sum_{j=1}^{|F|} (\tilde{\mathcal{S}}(A, \tilde{X}) - p_{i,j})^2, \tag{5.10}$$

where R is the number of nodes for training the shadow model, and $|F|$ is the number of class for nodes.

In particular, we explain why the MSE can be applied to guide the shadow model to mimic the behavior of the server model in VFGL. First, we give the definition of the decision boundary.

Definition 5.2 The decision boundary of a classifier \mathcal{V} can be expressed as follows:

$$\begin{aligned} B_{\mathcal{V}} : \quad & \mathcal{V}_l(x) - \mathcal{V}_{\hat{l}}(x) = 0 \\ \text{s.t. } & \hat{l} = \arg \max_l \mathcal{V}(x), \end{aligned} \quad (5.11)$$

where l is the l -th class of probabilities and x is the input of the classifier.

Then, the objective of the MSE can be described as

$$\begin{aligned} & \arg \min_{\tilde{\mathcal{S}}} MSE(\tilde{\mathcal{S}}(A, \tilde{X}), \mathcal{S}(h_{global})) \\ \Leftrightarrow & \arg \min_{\tilde{\mathcal{S}}} |\tilde{\mathcal{S}}(A, \tilde{X}) - \mathcal{S}(h_{global})| \\ \Leftrightarrow & \arg \min_{\tilde{\mathcal{S}}} |(\tilde{\mathcal{S}}_l(A, \tilde{X}) - \tilde{\mathcal{S}}_{\hat{l}}(A, \tilde{X})) \\ & - (\mathcal{S}_l(h_{global}) - \mathcal{S}_{\hat{l}}(h_{global}))| \\ \Leftrightarrow & \arg \min_{\tilde{\mathcal{S}}} |B_{\tilde{\mathcal{S}}} - B_{\mathcal{S}}|. \end{aligned} \quad (5.12)$$

Therefore, using MSE is helpful in fitting the shadow model to the server model. A shadow model that is more similar to the server model can provide more accurate information to the attackers.

5.4.3 Adversarial Attacks

The shadow model constructed in the above steps can be used for facilitating various centralized attackers to improve the attack success rate. In this chapter, we consider generating adversarial perturbations by adding/deleting edges in the graph, which is mainstream and effective. The adversarial embeddings for the target node v_t produced by the malicious client m are formulated as

$$\begin{aligned} \hat{h}_m^{v_t} &= f_{\theta}^m(\hat{A}, \tilde{X}, v_t) \\ \text{s.t. } & \hat{A} = ATK(A, \tilde{X}, \tilde{\mathcal{S}}), \end{aligned} \quad (5.13)$$

where $ATK(\cdot)$ is a kind of centralized attack method. And the perturbations are limited within the attack budget Δ , i.e., $\|\hat{A} - A\|_0 \leq 2 \cdot \Delta$. It should be noted that A and \hat{A} are symmetrical in the undirected graph.

Algorithm 5.1: Our method

Input: Original adjacency A , original node features X , target node v_t , starting epoch τ , train set V_{train} , attack budget Δ .

Output: The adversarial adjacency \hat{A} .

Train the VFGL until the starting epoch τ to start our method.

Divided the train set according to the node categories.

for $t = 1$ to $|V_{train}|$ **do**

- | Test the train examples according to Eqs. 5.7 and 5.8.
- | Obtain the neuron path P_t .

end

Count and select the target path P_T , and get the candidate nodes set C .

Identify the significant neuron k^1 by P_T , and select the top- M features and the target features set.

for c in C **do**

- | Modify the features of the candidate nodes following Eq. 5.9 and get \tilde{X} .

end

Train the VFGL with the manipulated features \tilde{X} continuously.

Query the server model and gain the probabilities p .

Establish the shadow model \tilde{S} with the local GNN and a L -layers MLP.

for $t=1$ to T_{shadow} **do**

- | Train \tilde{S} to minimize the MSE in Eq. 5.10.

end

Input the original adjacency, the manipulated features \tilde{X} and the shadow \tilde{S} into the centralized attack method.

for $i=1$ to Δ **do**

- | Generate the adversarial adjacency \hat{A} as Eq. 5.13.

end

return the adversarial adjacency matrix \hat{A} .

5.5 Experiments

In this section, we conduct experiments to evaluate the effectiveness of our method and research questions is to be answered: Can our method improve the attacks' performance significantly? What is the query cost of our method?

5.5.1 Experimental Settings

5.5.1.1 Datasets

The proposed method is evaluated on four real-world datasets: Cora [24], Cora_ML [24], Citeseer [24], and Pubmed [25], where Cora [24] contains 2,708 nodes, 5,429 edges, 1,433 features, 7 classes with average degree of 2.00. Cora_ML [24] contains 2,810 nodes, 7981 edges, 2879 features, and 7 classes with the average degree of 2.84. Citeseer [24] contains 3,327 nodes, 4,732 edges, 3,703 features, and 6 classes, average degree is 1.42. Pubmed [25] contains 19,717 nodes, 44,325

edges, 500 features, and 3 classes, the average degree is 2.25. To simulate the VFGL learning scenario where clients may not have the same data features, we divide the data and assign them to the clients. However, the graphs in these datasets are sparse. To avoid the appearance of isolated nodes after segmenting the graph, we randomly segment the features only while keeping the edges the same for every client. We test the performance of our method 5 times as well as other baselines and report the average results to eliminate the impact of the randomness.

5.5.1.2 Models and Parameter Settings

Three kinds of GNNs (i.e., GCN [23], SGC [26], GCNII [27]) are adopted as the local model in VFGL. For GCN, it is a 2-layer structure with a hidden size of 32, the output size of 16, and training period is 200. SGC has the same structure and parameters as GCN. The GCNII has 4-layer structure with a hidden size of [32, 32, 32], an output size of 16, and a training period of 1,000. The VFGL is trained using Adam with the learning rate set to 0.01. ReLU is adopted as the activation function for GCN and GCNII. For the malicious client, the scale γ for feature modification is searched in [0.01, 0.1], i.e., only $\gamma \cdot d$ (d is the number of node features) features are allowed to be modified per node. For the perturbation generator, the attack budget Δ is set to 1, considering the sparsity of the graph.

5.5.1.3 Evaluation Metrics

To measure the proposed method, attack success rate (ASR), contribution score (CS), and average queries (AQ) are adopted to evaluate the effectiveness and efficiency of our method.

- **CS:** Similar to [28], it is a metric to measure the contribution of each client during the training process. The CS of client i is defined as

$$\begin{aligned} CS_i &= \frac{con_i}{\sum_k^K con_k} \\ s.t. \quad con_i &= \sinh(\alpha \cdot \frac{acc_i}{\sum_k^K acc_k}), \end{aligned} \tag{5.14}$$

where acc_i is the training accuracy of the server model when only i 's data is input. α is the scaling constant of contribution. In this chapter, it is set to 5. K is the number of clients in the VFGL.

- **AQ:** The AQ is applied to measure the efficiency of the attack, which is defined as

$$AQ = \frac{1}{N_a} \sum_j^{N_a} q_j. \tag{5.15}$$

If the attack fails, the number of queries q is set to a default query budge Q . It should be pointed out that we take the number of times that a malicious client initiates a request to the server model as the number of queries. Frequent requests to the server in a short period of time are suspicious and inefficient. Therefore, the lower the AQ, the higher the efficiency of the attacker.

5.5.1.4 Baselines and Adversarial Attacks

Baselines. Due to the lack of research, the proposed method is compared with the other two methods we propose:

- **Random Features Attack (RFA):** It randomly selects the same number of features as our method for each candidate node, and the value is set to the largest value in the original feature distribution.
- **Specific Features Attack (SFA):** It randomly selects the same number of features as the specific features set. For each candidate node, the specific features are set to the largest value in the original feature distribution.

For a fair comparison, the number of candidate nodes is the same as our method.

Adversarial attacks. We consider four advanced centralized adversarial attacks as the attack generator. They are briefly described as follows:

- **FGA [6]:** It constructs the symmetrical edge gradient matrix of the original graph. Then, the maximal absolute edge gradient guides to generate perturbations iteratively.
- **GradArgmax [12]:** It calculates the gradient of edges based on the loss function, and the greedy algorithm is adopted to select the edges to be modified.
- **Nettack [10]:** It selects the candidate edges and node features by graph properties (e.g., degree distribution) and generates adversarial perturbations with the highest scoring edges or node features iteratively to fool the classifier.
- **SGA [8]:** It extracts a subgraph centered at the target node first. Then it generates adversarial attacks by flipping edges with the largest gradient in the subgraph.

To illustrate the query efficiency of our method, we select a query-based black-box adversarial attack method for comparison.

- **GeneticAlg [12]:** It adopts the genetic algorithm to generate the adversarial perturbations, which involves five major components, i.e., population, fitness, selection, crossover, and mutation. It has to query the target model to calculate the fitness score for each candidate solution that is decided by the population.

5.5.2 Effectiveness and Efficiency on Attacking VFGL

In this section, our method is conducted on four real-world datasets in two scenes, i.e., dual-client-based VFGL and multi-participant-based VFGL. Ablation and transferable experiments are also performed in this section. Besides, the proposed method is also compared with the query-based black-box attack regarding the number of queries needed.

5.5.2.1 Attack on Dual-Client-Based VFGL

As described above, in this scene, the node features are divided into two parts randomly. To verify the generality of our method, we conduct the experiments 5 times and report the ASR. The results are shown in Fig. 5.3, and the best attack performance is highlighted in bold. Besides, we visualize the neuron's output (Figs. 5.4 and 5.5) to illustrate the effect of our method on the local GNN. Some observations are concluded in this experiment.

(1) *Ours can significantly improve the performance of the centralized adversarial attacks and achieves SOTA attack performance compared with baselines.* Take FGA on GCN-based VFGL as an example, the ASR of the our method has reached more than 30% on Cora and Cora_ML and Citeseer, and even achieved a 40% attack performance on Cora_ML. Under the same settings, RFA and SFA can only achieve less than 30% attack performance improvement. As for Pubmed, the reason why attack performance is not obvious is that the attacks have already achieved a high ASR without using any data manipulation method, but our method still outperforms the baselines. Furthermore, we also note that RFA and SFA may impair attack performance due to their randomness (e.g., attack on GCN-based VFGL using GradArgmax).

(2) *Before and after using our method, the activation of neurons to training examples remains similar.* We visualized the output of neurons in the last layer of the local GNN for training examples of the same class. As shown in Fig. 5.4, it can be seen that in the case of normal training, the important neurons for these examples are #9 and #13, and the same after using our method. It indicates that our method does not introduce large differences on the training set, and it follows the original training process.

(3) *Neurons' activation to adversarial examples is changed significantly after using our method.* We visualize some nodes that cannot be attacked successfully under clean conditions (i.e., without data manipulation) but succeed under our method. In Fig. 5.5, for clean conditions, the perturbations at these nodes are not enough to change the activation of neurons, which leads to the failure of the attack. As for our method, compared with the clean case, the distribution of neurons' activation to benign examples is more concentrated in #9 and #13, which is similar to the case of the training set (Fig. 5.4). For adversarial examples, the distribution has changed significantly, and it is no longer #9 and #13. It explains why these nodes can be attacked successfully using our method from the perspective of neurons.

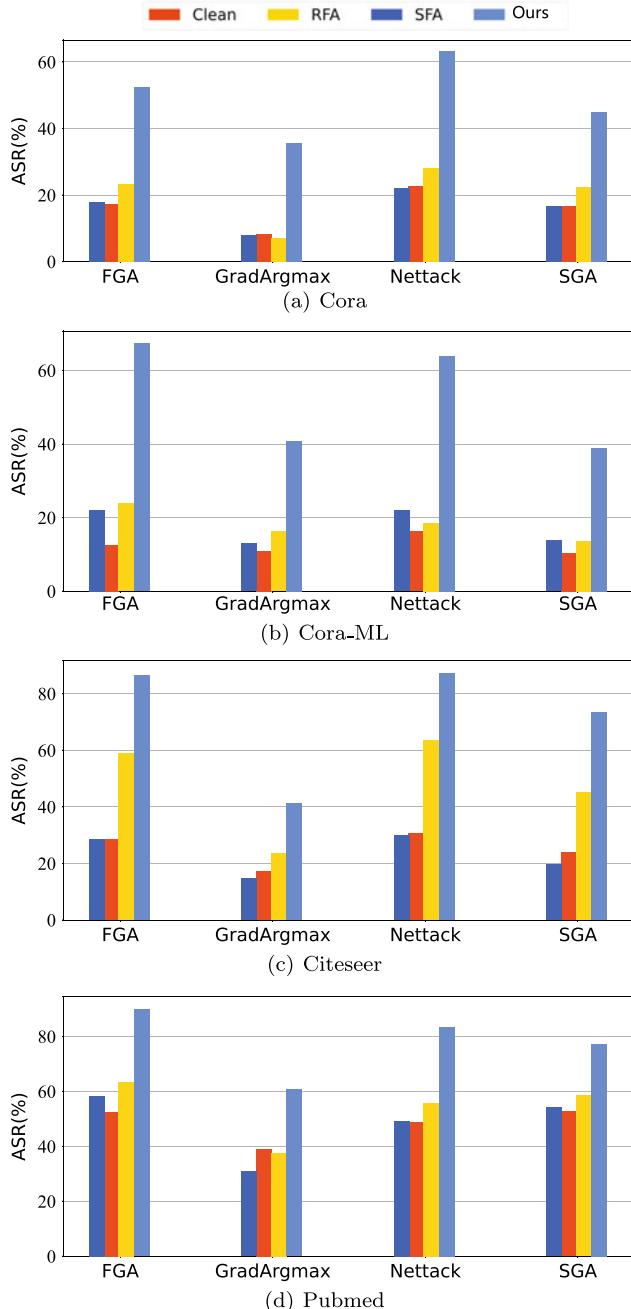


Fig. 5.3 The average attack success rate of four adversarial attacks on three kinds of local GNNs

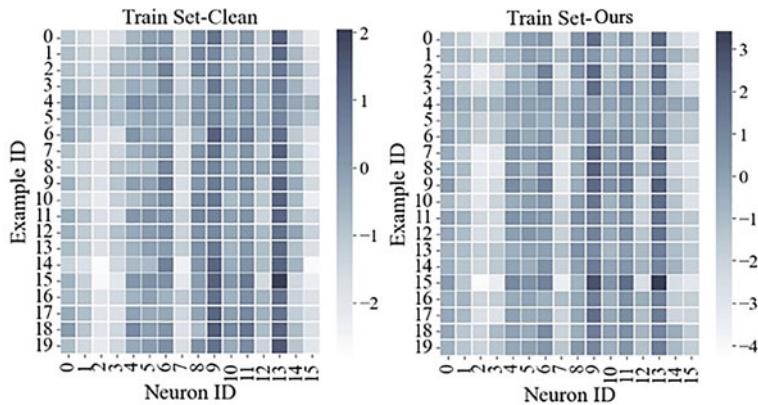


Fig. 5.4 The effect of our method on the neurons' activation of the training examples

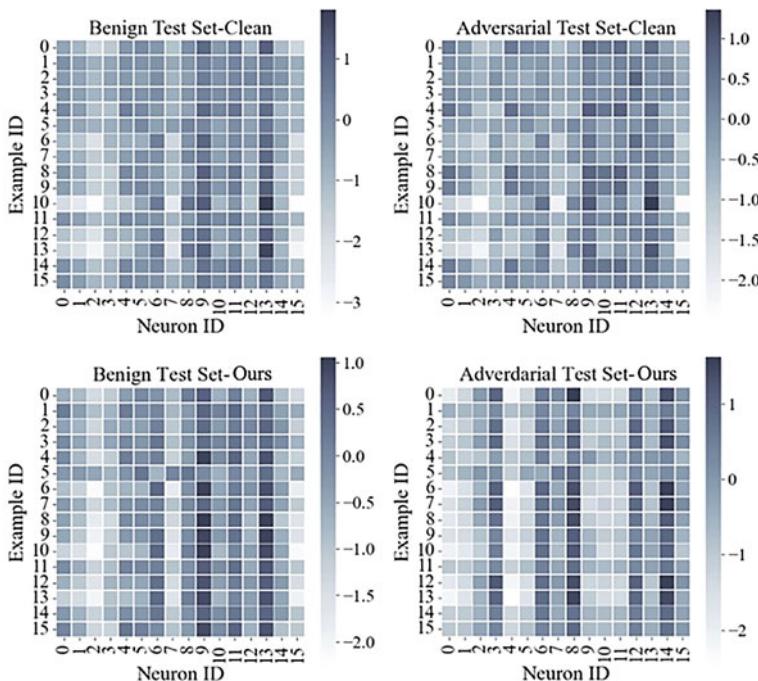


Fig. 5.5 The effect of our method on the neurons' activation of the benign examples and the adversarial examples

The above experiments suggest VFGL is vulnerable to adversarial attacks as well as the centralized GNNs. Besides, the centralized attacks transferable on VFGL are not as powerful as in the centralized scene. But the threat of these attacks can be improved significantly by adopting our method. Then, the sensitive neurons' identification shows that the activation shift of neurons is an important factor in attack success.

5.5.2.2 Attack on Multi-client-Based VFGL

Further, a more realistic scenario is considered to evaluate the threat of the proposed method, i.e., the multi-client-based VFGL. The number of the clients is set in [2, 4, 6, 8, 10]. The node features are split for each client and the structure of the graph is contained as well as the dual-client-based VFGL. The CS of the malicious client and the ASR of ours are reported in Table 5.1.

In general, the ASR of all attacks decreases as the number of clients increases, since the impact of adversarial perturbations by the malicious client will gradually decrease. In other words, when the number of clients is large enough, the perturbations of the malicious client will not be enough to confuse the server model. Therefore, improving the contribution of the malicious client to the server model, which makes the server model more dependent on the malicious client's data, is a feasible measure to improve the attack's capability in the case of multiple clients. It

Table 5.1 ASR and CS of ours with different number of clients in GCN-based VFGL

Dataset	Method	Number of clients									
		2		4		6		8		10	
		ASR (%)	CS	ASR (%)	CS	ASR (%)	CS	ASR (%)	CS	ASR (%)	CS
Cora	Clean	17.581	0.418	9.251	0.403	6.922	0.155	2.572	0.214	5.021	0.067
	RFA	23.972	0.529	10.855	0.419	10.504	0.156	2.313	0.241	5.022	0.110
	SFA	17.581	0.381	7.131	0.398	5.457	0.140	2.181	0.215	5.023	0.057
	Ours	47.374	0.626	38.643	0.483	27.799	0.272	25.440	0.316	27.352	0.126
Cora_ML	Clean	16.132	0.477	3.492	0.248	6.721	0.170	4.883	0.155	4.309	0.129
	RFA	32.956	0.645	12.693	0.312	10.222	0.212	11.234	0.202	5.578	0.140
	SFA	21.848	0.464	9.204	0.222	5.147	0.175	7.103	0.155	4.308	0.135
	Ours	57.711	0.508	42.222	0.344	38.786	0.255	34.081	0.298	13.197	0.209
Citeseer	Clean	23.497	0.494	10.453	0.262	12.22	0.241	9.344	0.150	9.426	0.168
	RFA	48.045	0.531	10.454	0.311	19.843	0.272	12.309	0.211	14.505	0.226
	SFA	27.306	0.436	22.305	0.254	9.265	0.150	6.382	0.150	11.964	0.164
	Ours	78.097	0.632	54.052	0.480	40.587	0.273	33.471	0.207	26.780	0.297
Pubmed	Clean	65.117	0.593	51.448	0.313	15.341	0.177	53.680	0.194	1.253	0.089
	RFA	69.196	0.577	53.488	0.324	15.851	0.177	50.625	0.183	1.254	0.084
	SFA	64.607	0.556	49.916	0.309	12.28	0.167	52.660	0.184	1.253	0.089
	Ours	88.063	0.773	67.255	0.346	64.804	0.209	81.724	0.295	28.790	0.116

can be seen that our method improves the CS of the malicious client and outperforms baselines in general, corresponding to a higher ASR. It is worth noting that in the case of two clients on Cora_ML, although RFA gains a higher CS, our performance is still better than RFA-SGA. We believe that it is because the shadow model can be better trained with the data modified by our method and provides better guidance for attacks. In addition, it can be observed that in the case of eight clients on Pubmed, the performance is better than the case of six clients, which benefits from the higher CS.

In summary, in the multiple client cases, the ASR of attackers shows a downward trend with the increasing number of clients. It is caused by the declining influence of the malicious client. Compared with RFA and SFA, the effect of our method on the improvement of CS is more obvious, so it can improve ASR more significantly.

5.5.2.3 Transferability and Ablation of Our Method

The transferability and ablation of our method are evaluated in this subsection. There are nine combinations (i.e., three local models and three shadow models in our experimental configuration) for a given dataset. Besides, to verify the effectiveness of the MSE, the cross-entropy loss (CE) calculated from the hard labels of the training set is adopted as a comparison. The experiments are conducted on the Cora dataset and Citeseer dataset. FGA is applied as the attack generator and ASR is used as the metric. The results are shown in Fig. 5.6.

Transferability: Take ours-MSE as an example, for GCN and SGC, our method has a strong transfer attack ability, that is, the shadow model established by GCN/SGC can effectively attack SGC/GCN, due to their similar structure. For GCNII, the transfer attack ability drops subtly, which can be caused by the large structural difference between GCNII and GCN (SGC) and it is difficult to establish a fully consistent

			Clean-CE			Ours-CE			Clean-MSE			Ours-MSE			
Cora	Shadow Model		GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN
	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC
	11%	13%	8%	62%	45%	29%	14%	13%	34%	65%	61%	68%	68%	70%	75%
Citeseer	Shadow Model		GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN
	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC
	26%	25%	13%	87%	66%	63%	23%	28%	19%	94%	69%	65%	93%	74%	52%
Local Model	GCN		GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN
	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC	GCN	GCNII	SGC
	27%	25%	18%	91%	74%	48%	23%	26%	27%	85%	64%	67%	85%	64%	67%
Local Model	GCNII		GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII
	GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII	GCN	SGC	GCNII	GCN	SGC
	14%	11%	37%	84%	56%	56%	21%	19%	17%	94%	69%	65%	93%	74%	52%

Fig. 5.6 Transferability and ablation of our method. We present the attack performance (ASR) of nine combinations of the local GNN and the shadow model

shadow model. Compared with the Clean-MSE, in general, ours-MSE demonstrates stronger transferability (at least 50% ASR is exhibited on both datasets, while this is only 34% at best under Clean-MSE conditions). Similar conclusions can also be obtained in ours-CE.

Ablation: The proposed method consists of two important components, i.e., the data manipulation and the MSE-based shadow model construction. In the ablation experiments, we use these two components as ablation objects.

Take the most general case (i.e., no data manipulation and CE is adopted as the loss function of the shadow model) as the base. Some observations are summarized as follows.

(1) *The data manipulation enhances the threat of the attack.* As shown in Fig. 5.6, when data manipulation and CE are applied, the ASR is greatly improved in any combination. It is improved from 8% to 29% even in the combination of GCN and GCNII with different structures. We believe this is due to the increased contribution of malicious parties caused by data manipulation. As a result, the perturbation has a greater impact on the server model.

(2) *Using an MSE-based shadow model enhances the transferability of the attack.* On the Cora dataset, using GCN as the shadow model to attack, the ASR increases from 8 to 34%. It is important to note that while the MSE is adopted without data manipulation, the performance of the attack is slightly improved. This phenomenon is more apparent in the transfer attack. In addition, the MSE can effectively utilize the behavioral information leaked by probabilities between models with large structural differences, which is more conducive to the establishment of the shadow model. To better illustrate it, taking the Pubmed dataset as an example, we project the embeddings extracted from the server model and the shadow model by using t-SNE, which is shown in Fig. 5.7. It can be observed from the distribution of t-SNE

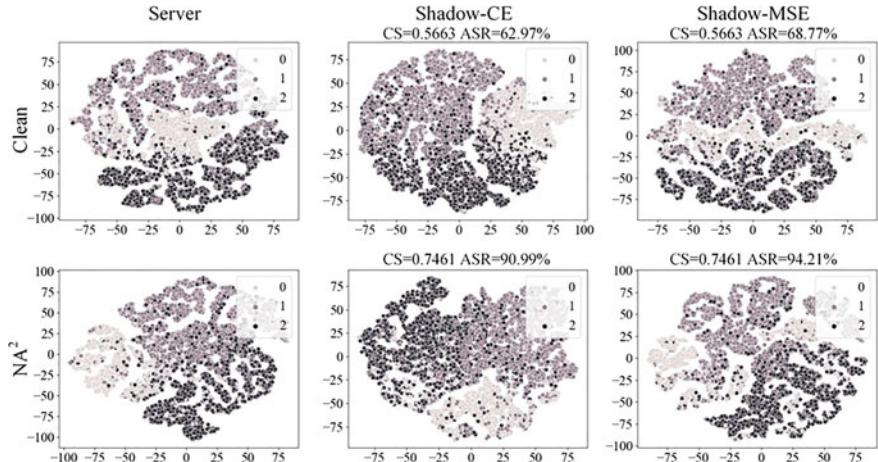


Fig. 5.7 The embeddings which obtained from the server model and the shadow model in the GCN-based VFGL on the Pubmed dataset. They are projected into a two-dimensional space using t-SNE. The CS and ASR of SGA are reported at the same time

that the distribution using CE is significantly different from the target distribution (Server). It shows that CE only focuses on performance and ignores server model behavior. The t-SNE distribution extracted from the shadow model built with our method using MSE is most similar to the server model with our method. Besides, the CS and ASR (SGA) of the shadow model with ours-MSE show the best attack performance among all the combinations.

In summary, combining the advantages of the above two components (i.e., the data manipulation and the MSE-based shadow model), ours-MSE achieves SOTA performance in both ASR and transfer attack ability.

5.5.2.4 Efficiency of Attacking VFGL

In order to verify the query efficiency of our method, we compare ours-FGA with a query-based black-box attack method, namely, GeneticAlg. GeneticAlg is a genetic algorithm-based attack method. The population of GeneticAlg is set to 20, and the number of iterations is set to 10. Thus, the maximum number of queries for a target node is 200. If the attack fails, the number of queries is fixed to 200. Figure 5.8 reports the ASR and AQ of ours-FGA and GeneticAlg on four datasets and three local GNNs.

Since ours-FGA only needs to initiate one request to the server model to get the probabilities, the number of queries for ours-FGA is always fixed to 1. As for GeneticAlg, more queries are necessary to optimize the perturbations. Therefore, GeneticAlg has far more queries than ours-FGA. Besides, in each setting, the ASR of ours-FGA far exceeds that of GeneticAlg, which shows the superiority of ours-FGA in attack performance and query efficiency.

5.6 Conclusion

Our proposed framework is an adversarial attack system that is highly efficient in terms of queries. The proposed method locates key neurons via the key neuron path and enhances the corresponding node features to mitigate malicious client contributions. A shadow model is established using the manipulated data and probabilities from the server model. Extensive experiments demonstrate that our method can significantly improve the performance of centralized adversarial attacks against VFGL, achieving SOTA performance. Despite implementing a defense mechanism, our method continues to achieve impressive results in their attacks. In addition, t-SNE is used to identify and visualize sensitive neurons, providing insight into the effectiveness of our method.

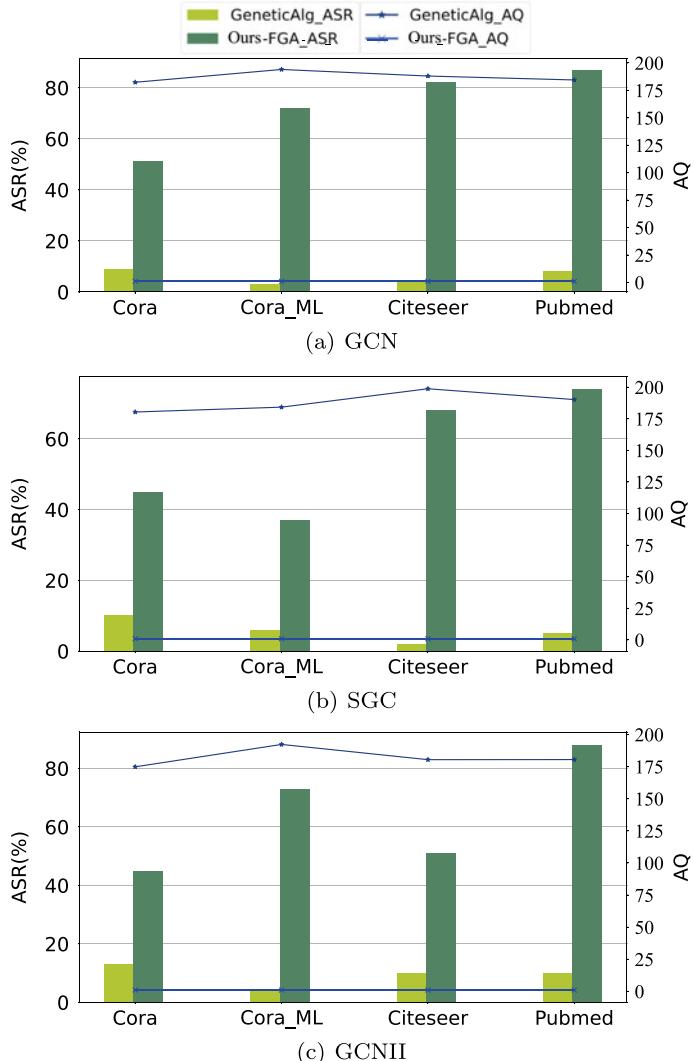


Fig. 5.8 Comparison of attack success rate and the average number of queries between ours-FGA and query-based black-box attack method. There are 100 examples randomly selected for the evaluation

References

1. Zheng, L., Zhou, J., Chen, C., Wu, B., Wang, L., Zhang, B.: Asfgnn: automated separated-federated graph neural network. *Peer-To-Peer Netw. Appl.* **14**(3), 1692–1704 (2021)
2. Wang, B., Li, A., Li, H., Chen, Y.: Graphfl: a federated learning framework for semi-supervised node classification on graphs (2020). [arXiv:2012.04187](https://arxiv.org/abs/2012.04187)
3. He, C., Balasubramanian, K., Ceyani, E., Yang, C., Xie, H., Sun, L., He, L., Yang, L., Yu, P.S., Rong, Y., et al.: Fedgraphnn: a federated learning system and benchmark for graph neural networks (2021). [arXiv:2104.07145](https://arxiv.org/abs/2104.07145)
4. Zhou, J., Chen, C., Zheng, L., Wu, H., Wu, J., Zheng, X., Wu, B., Liu, Z., Wang, L.: Vertically federated graph neural network for privacy-preserving node classification (2020). [arXiv:2005.11903](https://arxiv.org/abs/2005.11903)
5. Ni, X., Xu, X., Lyu, L., Meng, C., Wang, W.: A vertical federated learning framework for graph convolutional network (2021). [arXiv:2106.11593](https://arxiv.org/abs/2106.11593)
6. Chen, J., Wu, Y., Xu, X., Chen, Y., Zheng, H., Xuan, Q.: Fast gradient attack on network embedding (2018). [arXiv:1809.02797](https://arxiv.org/abs/1809.02797)
7. Chen, J., Zhang, J., Chen, Z., Du, M., Xuan, Q.: Time-aware gradient attack on dynamic network link prediction. *IEEE Trans. Knowl. Data Eng.* (2021)
8. Li, J., Xie, T., Liang, C., Xie, F., He, X., Zheng, Z.: Adversarial attack on large scale graph. *IEEE Trans. Knowl. Data Eng.* (2021)
9. Wu, H., Wang, C., Tsyshetskiy, Y., Doherty, A., Lu, K., Zhu, L.: Adversarial examples for graph data: deep insights into attack and defense. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 4816–4823 (2019)
10. Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial attacks on neural networks for graph data. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2847–2856 (2018)
11. Zügner, D., Günnemann, S.: Adversarial attacks on graph neural networks via meta learning (2019). [arXiv:1902.08412](https://arxiv.org/abs/1902.08412)
12. Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L.: Adversarial attack on graph structured data. In: International Conference on Machine Learning, pp. 1115–1124. PMLR (2018)
13. Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Wang, X., Zhu, W., Huang, J.: Adversarial attack framework on graph embedding models with limited knowledge. *IEEE Trans. Knowl. Data Eng.* (2022)
14. Ma, J., Ding, S., Mei, Q.: Towards more practical adversarial attacks on graph neural networks. *Adv. Neural. Inf. Process. Syst.* **33**, 4756–4766 (2020)
15. Ma, Y., Wang, S., Derr, T., Wu, L., Tang, J.: Graph adversarial attack via rewiring. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 1161–1169 (2021)
16. Qiu, Y., Leng, J., Guo, C., Chen, Q., Li, C., Guo, M., Zhu, Y.: Adversarial defense through network profiling based path extraction. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 4777–4786 (2019)
17. Zang, X., Xie, Y., Chen, J., Yuan, B.: Graph universal adversarial attacks: A few bad actors ruin graph learning models (2020). [arXiv:2002.04784](https://arxiv.org/abs/2002.04784)
18. Wang, X., Cheng, M., Eaton, J., Hsieh, C.J., Wu, F.: Attack graph convolutional networks by adding fake nodes (2018). [arXiv:1810.10751](https://arxiv.org/abs/1810.10751)
19. Sun, Y., Wang, S., Tang, X., Hsieh, T.Y., Honavar, V.: Node injection attacks on graphs via reinforcement learning (2019). [arXiv:1909.06543](https://arxiv.org/abs/1909.06543)
20. Zou, X., Zheng, Q., Dong, Y., Guan, X., Kharlamov, E., Lu, J., Tang, J.: Tdgia: Effective injection attacks on graph neural networks. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 2461–2471 (2021)
21. Tao, S., Cao, Q., Shen, H., Huang, J., Wu, Y., Cheng, X.: Single node injection attack against graph neural networks. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 1794–1803 (2021)

22. Zhang, H., Shen, T., Wu, F., Yin, M., Yang, H., Wu, C.: Federated graph learning—a position paper (2021). [arXiv:2105.11099](https://arxiv.org/abs/2105.11099)
23. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks (2016). [arXiv:1609.02907](https://arxiv.org/abs/1609.02907)
24. McCallum, A.K., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. *Inf. Retr.* **3**(2), 127–163 (2000)
25. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. *AI Mag.* **29**(3), 93 (2008)
26. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International Conference on Machine Learning, pp. 6861–6871. PMLR (2019)
27. Chen, M., Wei, Z., Huang, Z., Ding, B., Li, Y.: Simple and deep graph convolutional networks. In: International Conference on Machine Learning, pp. 1725–1735. PMLR (2020)
28. Lyu, L., Xu, X., Wang, Q., Yu, H.: Collaborative fairness in federated learning. In: Federated Learning, pp. 189–204. Springer, Berlin (2020)

Chapter 6

Targeted Label Adversarial Attack on Graph Embedding



6.1 Introduction

The popularity of graph embedding methods has led to an increased focus on their security vulnerabilities. The need to safeguard personal privacy and address concerns surrounding extensive graph mining has sparked research into adversarial attacks [1–8] on graph embedding methods. Based on diverse modification approaches, attacks on graph-structured data can be primarily divided into graph structure attacks [1–3, 5, 8], feature attacks [1, 7] and graph injection attacks [4, 6]. Given that graph structure information typically holds greater significance in graph embedding [1, 5], current studies focus more on graph structure attacks. In a more specific manner, attackers initially acquire the perturbations candidate using various optimizers such as gradient learning [3], evolutionary computing [9], and meta-learning [10]. Subsequently, the perturbations with the highest influence on graph analysis tasks are chosen from the pool of candidates. In this study, our focus lies on graph structure attacks specifically targeting node classification.

Typically, the objective of the attacker is to manipulate the model in such a way that it predicts the expected label (label \equiv category) for the target node which aligns more closely with the demands of practical scenarios. For example, on a shopping platform, a user who is annoyed by excessive advertisements may disguise themselves as a particular user (whose label indicates interest in a specific item) by modifying their attributes or establishing connections with specific users. The targeted label attack technique enables the user to evade the recommendation system's detection of their privacy preferences. Furthermore, the user can receive information related to the interests they are willing to reveal, rather than the items they are uninterested in. However, the current studies chiefly concentrate on getting an inaccurate prediction outcome for the target example [1–3] or decreasing the prediction accuracy of the overall graph analysis model [10] as much as possible, rather than the targeted label attack. This means that even though privacy is being protected through existing methods, users are still troubled by unnecessary or excessive information.

Furthermore, as the robustness enhancement of graph embedding methods [5, 11–17] improves, improving the concealment of adversarial examples to evade defense and detection methods is also a pressing issue that needs to be addressed. Several works [5, 15, 17] point out that adding edges between dissimilar nodes is key to an effective attack on the graph. Take the two classic adversarial methods, NETTACK [1] and FGA [3], as examples. NETTACK selects candidate edges and node features based on graph properties and generates adversarial attacks using two score functions iteratively to deceive the classifier. On the other hand, FGA modifies the edges in the graph using the maximal absolute edge gradient. As shown in Fig. 6.1a, NETTACK and FGA have a preference for adding perturbations between more dissimilar nodes. Similarly, other attack methods also encounter the same issue as their main objective is to generate perturbations that have the maximum impact on the prediction confidence of target examples. Consequently, they tend to overlook the similarity between connected nodes. Figure 6.1b demonstrates the defense method that relies on the similarity between nodes. The defense methods [5, 15, 17] effectively protect against the majority of attack methods by employing the fundamental concept of eliminating edges that connect nodes with significant dissimilarity. This suggests that the stealthiness of graph adversarial perturbations at the level of node similarity should also be taken into consideration.

In conclusion, while the current adversarial attacks on graph embedding methods have demonstrated satisfactory attack performance, they still face challenges in two key aspects: (i) The targeted label attack. Previous research primarily focuses on diminishing the performance of target graph embedding methods, neglecting the potential application prospects of targeted label attacks. (ii) Stealthiness. Many of

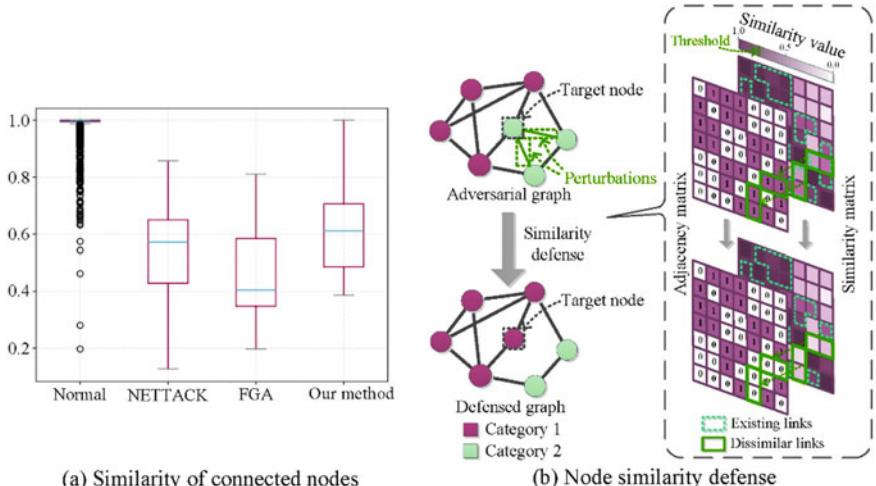


Fig. 6.1 An illustrative example on the distinction between perturbations and normal edges on Cora, and the node similarity defense method. Compared with NETTACK and FGA, the perturbations generated by our method are stealthier in node similarity level

the current attack methods prioritize attack performance over stealthiness, making them vulnerable to defense techniques based on node similarity. Therefore, a significant challenge lies in improving the stealthiness of attacks while maintaining their effectiveness, which remains an open problem to be addressed.

As mentioned above, it remains a challenge to bring up an attack method that considers both the targeted label attack and the stealthiness of the perturbations. In the computer vision area, Mohsen et al. [18] generated minimum perturbations that are sufficient to change the classification label based on an iterative linearization of the classifier. Drawing inspiration from this concept, we introduce a method that aims to achieve a precise targeted label attack with minimal perturbations. In our method, we generate adversarial perturbations by determining the minimum distance from the instance to the constructed classification boundary of the target model. By minimizing the change in prediction confidence for the target instance, our method enhances the stealthiness of perturbations at the node similarity level. To summarize, the key contributions of our work can be outlined as follows:

- We propose a targeted label adversarial attack on graph embedding methods. It aims to achieve a precise and targeted label attack with perturbations that are imperceptible. This is accomplished by reversing the optimization procedure of the target graph embedding model, which is based on the classification boundary.
- About the stealthiness of our method attack and the processing power of the local network, we propose a perturbation-limited attack to limit the perturbation scale. Meanwhile, it can reduce the complexity of our method.
- Extensive experiments demonstrate that our method achieves state-of-the-art attack performance in a stealthier manner at the node similarity level.

6.2 Related Work

The related work can be generalized into several categories, graph embedding methods, and graph attacks.

6.2.1 *Graph Embedding Methods*

Recently, many studies have focused on embedding graph into a low-dimensional vector space based on the word2vec model [19]. They are called shallow graph embedding, such as DeepWalk [20], Node2vec [21], LINE [22], and GraRep [23], DeepWalk [20] is the pioneering approach to learn language from a graph. It adopts a random walk technique to sample a sequence for every node in the graph. These generated sequences are then treated as sentences and fed into the skip-gram mechanism for language learning. Tang et al. [22] proposed a novel graph embedding method

called LINE. LINE can be considered as a special case of DeepWalk, where the window size of contexts is specifically set to one. Taking inspiration from DeepWalk, Grover et al. [21] proposed an extension of DeepWalk, called Node2vec. To enhance the flexibility in generating context nodes, Node2vec utilizes a biased second-order random walk model. Cao et al. [23] introduced a novel embedding method called GraRep. GraRep aims to preserve the proximity between nodes in a graph by constructing multiple k-step probability transition matrices.

Furthermore, lots of deep embedding methods [24–26] are proposed as well, which are typically built upon deep learning models, such as convolutional neural network (CNN) [27, 28] and generative adversarial network (GAN) [29]. Kipf et al. [24] introduced Graph Convolutional Networks (GCN) for the task of semi-supervised node classification. GCN has the advantage of scaling linearly with the number of graph nodes, making it efficient for large-scale graphs. It learns hidden layer representations by encoding both the local graph structure and the features of nodes. Similarly, Pham et al. [26] proposed column network (CLN), which is a deep learning model for collective classification. Compared with GCN, CLN emphasizes relation learning, which can process multi-relational data. Monti et al. proposed a unified framework, MoNet [30], which generalizes the convolution to non-Euclidean domains. To enhance the extraction of critical information, Wang et al. [31] proposed a non-local neural network, which is capable of capturing more detailed information on graph structure. Wang et al. [32] proposed GraphGAN, which integrates two categories of graph representation learning techniques.

6.2.2 *Graph Attacks*

Existing attack methods have developed various optimization algorithms to generate impactful perturbations for the target graph embedding model. Based on different modification strategies, these attack methods primarily concentrate on graph structure attacks, graph feature attacks, and graph injection attacks.

Graph structure attacks. Zügner et al. [1] proposed NETTACK, which is the first adversarial attack on the graph. It generates the adversarial graph by leveraging score functions under a constrained budget. Zügner et al. [10] further proposed Meta-Self, which regards the graph as an optimistic hyperparameter and employs meta-gradient optimization without prior knowledge of the classification model or its training weights. Dai et al. [2] proposed RL-S2V, they altered the graph structure using reinforcement learning, incorporating prediction feedback from the target classifier. Chen et al. [3] proposed fast gradient attack (FGA), it extracts the gradient of pairwise nodes and subsequently selects the pair of nodes with the highest absolute edge gradient to update the adversarial graph. To mitigate the error induced by the gradient information on discrete graph data, Wu et al. [5] proposed an integral gradient to accurately represent the impact of perturbing specific edges while still leveraging the advantages of parallel computations. Considering the attack in black-box scenario, GF-Attack [8] considers the graph embedding model as the new graph

signal generated by the graph filter and feature transformation, attacking the graph filter instead of the graph embedding model.

Graph feature attacks. While graph feature attacks may be less effective compared to graph structure attacks, they enrich the attack strategies with different requirements. For feature attacks, Avishek et al. [33] considered adversarial attacks as a generative problem. They proposed an encoder–decoder framework (DAGAER) to create node features with minor perturbations. To accomplish a more subtle attack, Takahashi et al. [7] proposed POISONPROBE, which attacks the node features of the target node’s two-hop or even multi-hop neighbor nodes.

Graph injection attacks. Graph injection attacks are designed to tackle situations where modifying the existing graph structure or features is not feasible. This can occur when the attacker lacks the necessary permissions to modify the underlying graph database. Wang et al. [4] connected fake nodes to existing nodes based on a greedy algorithm, and designed a discriminator to ensure the perturbations remain stealthy. Besides, NIPA [6] performs the fake node attack by training deep reinforcement learning agents based on reinforcement learning. TDGIA [34] connects the smooth feature optimization-based nodes with the original nodes using the topological defective edge selection strategy.

In most attack methods, the optimization objective is to identify perturbations that maximize the difference between the predicted confidence and the ground truth, making it challenging to accurately misclassify instances into the desired target label. Additionally, these methods may introduce unnecessary and excessive changes in the prediction confidence, which can be easily detected by certain defense mechanisms.

6.3 Preliminaries

First, we define notations which are used throughout the chapter. A graph can be represented by $G = \{V, E, X\}$, where V is the node set with $|V| = N$ and E is the edge set. $X \in \mathcal{R}^{N \times C}$ is the node attribute matrix, where C denotes the dimension of X . Generally, the adjacency matrix A contains the information of V and E , so we use $G = (A, X)$ to represent a graph more concisely.

6.3.1 GCN Model

Indeed, Graph Convolutional Networks (GCNs) are a popular application of traditional convolutional neural networks in the graph domain. GCNs utilize an efficient layer-wise propagation rule, which is based on the first-order approximation of spectral convolutions on graphs. This approach has demonstrated satisfactory performance in semi-supervised node classification tasks. Lots of graph attacks [1, 3, 10, 35] have demonstrated that adversarial graphs generated using GCN as the target model exhibit excellent transferability. This means that these adversarial graphs can

also achieve effective attack performance against other graph embedding algorithms. Hence, in our approach, we choose GCN as the target classifier to generate adversarial graphs. However, it is important to note that we are not limited to GCN and can potentially target and deceive various other graph embedding algorithms as well.

In specific, we employ a two-layer GCN model with SoftMax classifier, which can be defined as

$$Z = \text{softmax}(\tilde{A}\sigma(\tilde{A}XW_0)W_1), \quad (6.1)$$

where $\tilde{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$. A is the adjacency matrix, and $\tilde{A} = A + I_N$ is the adjacency matrix of the graph with self-connections. $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ denotes the degree matrix of \tilde{A} . $W_0 \in \mathcal{R}^{C \times H}$ is the input-to-hidden weight matrix with the hidden layer of H feature maps, and $W_1 \in \mathcal{R}^{H \times |F|}$ is the hidden-to-output weight matrix. The value of H determines the quality of the learned low-dimensional representation Z , and it is usually selected based on experimental results. σ denotes the ReLU active function.

The loss function is defined as the cross-entropy error over all labeled nodes.

$$L = -\sum_{l=1}^{|V_L|} \sum_{k=1}^{|F|} Y_{lk} \ln(Z_{lk}), \quad (6.2)$$

where V_L is the training node set with labels. $|F|$ is the dimension of the low-dimensional representation Z , which is equal to the number of categories. Y is the real label confidence list with $Y_{lk} = 1$ if node v_l belongs to category τ_k and $Y_{lk} = 0$ otherwise. $Z_{lk} = 1$ denotes the predicted probability that node v_l belongs to τ_k .

In the training process, the GCN model uses the classical gradient descent to optimize the parameters.

$$W_i^{m+1} = W_i^m - \eta \frac{\partial L}{\partial W_i^m}, \quad (6.3)$$

where η is the learning rate. During each iteration, the weights W_i ($i \in \{0, 1\}$) are updated.

6.4 Methodology

Based on the graph embedding algorithms applied to the node classification task, our method is designed to achieve targeted label attack by adding or removing a few edges of the original graph.

6.4.1 Main Framework

In this section, we take GCN as the target graph embedding algorithm as an example to help us introduce the attack process of our method. Our method consists of two parts, which are adversarial graph generation and adversarial attack, respectively. The framework of our method is shown in Fig. 6.2.

- **Adversarial graph generation:** For the trained GCN model, we use an iterative linearization method to generate the minimum adjacency matrix perturbations that are sufficient to change the classification result of the attacked node. The final adversarial graph is obtained by mixing the perturbations and the original adjacency matrix.
- **Adversarial attack:** We use the generated adversarial example to mislead the classification results of the attacked node. Since GCN can accurately extract the hidden representation of nodes from the graph structure and node attributes, the perturbations generated by the GCN as the target classifier are universal, and the attack has strong transferability.

As shown in Fig. 6.2, for a graph dataset, we first derive the node classification results based on the adjacency matrix and the trained GCN model in the adversarial graph generation stage. Then, we calculate the classification boundaries (f_1 , f_2 , and f_3) for the attacked node v_{tar} . We can also derive the minimum distance d_1 , d_2 , and d_3 , where $d_1 > d_3 > d_2$. In untargeted label attack, we select the most vulnerable label whose classification boundary (f_2) is closest to v_{tar} . Then the perturbation matrix is calculated, and the adversarial graph can be derived by adding/deleting the edges.

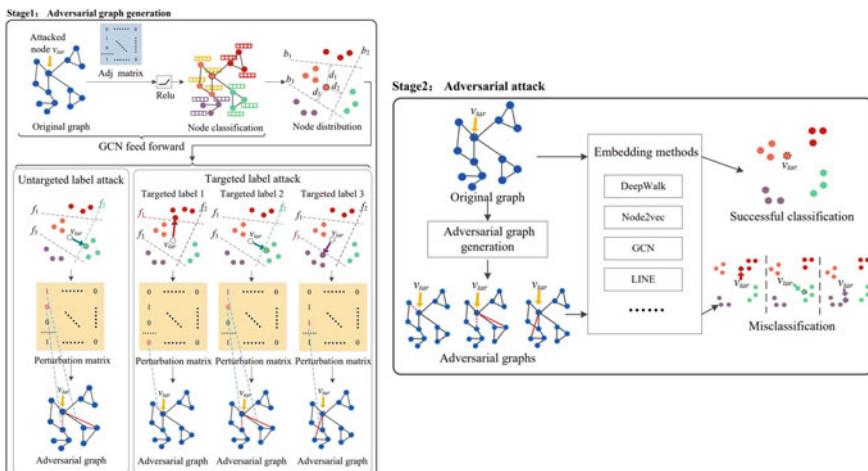


Fig. 6.2 The framework of our method

In the targeted label attack, the only distinction is that we assign the target label to the attacked node. Finally, we employ the adversarial graphs to assess the impact of our method on different network embedding algorithms during the adversarial attack stage.

6.4.2 Adversarial Graph Generation

In Sect. 6.3, we introduce the structure and training processing of a two-layered GCN. Our approach involves utilizing GCN as the target node classifier and generating adversarial graphs using our method. The pseudocode of adversarial graph generation is given in Algorithm 6.1.

6.4.2.1 Perturbations of Adjacent Matrix

For a trained node classifier, it usually learns a $|F|$ -dimensional low-dimensional representation from a given graph structure and node attributes, where $|F|$ is the number of node categories. We can define the node classifier as $f : R^C \rightarrow R^F$ (when the target classifier is GCN, Eq. 6.1 can represent f). For node v_i , we use the following mapping equation to describe a node classifier:

$$\hat{k}(x_i, A) = \arg \max_k f_k(x_i, A), \quad (6.4)$$

where \hat{k} denotes the predicted label of the node v_i in the classifier f , x_i is the attribute vector of node v_i , A is the graph adjacency matrix, and $f_k(x_i, A)$ is the output of the k -th category of $f(x_i, A)$.

Based on Eq. 6.4, if we aim to influence the node classification result of node v_i , we can change x_i or A . It is important to note that altering the edges in a graph affects all dimensions of node features during aggregation [5] while modifying features only affects partial features, and the node features may be continuous, making it challenging to control the attack within a given budget [36]. Moreover, in realistic systems such as transaction networks, there are limitations imposed by permissions. It is often easier to generate transaction records by transferring money between accounts rather than tampering with the attributes of the target user. Considering these factors, our method conducts attacks by altering the adjacency matrix of the graph instead of modifying node attributes. Equation 6.4 can be modified as

$$\hat{k}_i(A) = \arg \max_k f_k^i(A). \quad (6.5)$$

Most existing attack methods aim to find perturbations that maximize the impact on the prediction confidence of the target node. However, this can lead to excessive and unnecessary deviations in the prediction confidence. Additionally, these pertur-

bations tend to occur between more dissimilar nodes, making them easier to detect by various defense methods [5, 15, 17]. Indeed, constructing classification boundaries can be helpful in identifying perturbations that result in a more precise change in the prediction confidence of the target instance. By finding the perturbations that are just right, we can potentially avoid unnecessary deviations in the prediction confidence. This approach can lead to more effective and subtle attacks that are harder to detect by defense methods.

Our method on a simplified linear classifier. Considering that the general node classifiers are mostly complex non-linear classifiers, we first simplify the problem and introduce how to find the classification boundary and the minimum perturbations for the linear classifier so as to enhance our comprehension of the attack process employed by our method. We assume a linear classifier as $f^i(A) = A W_{\text{liner}} + b_{\text{liner}}$, where $W_{\text{liner}} \in \mathcal{R}^{N \times |F|}$ and $b_{\text{liner}} \in \mathcal{R}^{N \times 1}$ are trained classifier parameters. For the node v_i , the prediction of the classifier can be described as

$$\forall k : Aw_k + b_k < Aw_{\hat{k}_i(A)} + b_{\hat{k}_i(A)} \quad (k \neq \hat{k}_i(A)). \quad (6.6)$$

It means that for a well-trained classifier, the prediction probability corresponding to the true category of v_i should be greater than the other categories. The objective of the attack is to intentionally misclassify the target node by adding minimum perturbations \mathcal{P} on A . Therefore, the attack problem can be modeled as

$$\begin{aligned} & \arg \min \|\mathcal{P}\|_2 \\ \text{s.t. } & \exists k : (A + \mathcal{P})w_k + b_k \geq (A + \mathcal{P})w_{\hat{k}_i(A)} + b_{\hat{k}_i(A)}, \end{aligned} \quad (6.7)$$

where \mathcal{P} is perturbation matrix, and its calculation process will be introduced later. $w_k(w_{\hat{k}_i(A)})$ and $b_k(b_{\hat{k}_i(A)})$ are the $k(\hat{k}_i(A))$ -th column of W_{liner} and b_{liner} , respectively. Equation 6.7 indicates that when the well-designed perturbations are added to the input, the model will predict the example as k instead of $\hat{k}_i(A)$. Besides, Eq. 6.7 indicates that our method aims to find the minimum perturbation \mathcal{P} , so that the target classifier $f^i(\cdot)$ can recognize v_i as the k -th category based on the adversarial graph $(A + \mathcal{P})$ with greater probability than the $\hat{k}_i(A)$ -th category. As shown in stage 1 of Fig. 6.2, we hope that the attacked node v_{tar} can just be classified as the wrong k -th category. Calculating the shortest distance from v_{tar} to the classification boundary of the k -th category is the most straightforward manner. The classification boundary between the specified k -th category and the initial $\hat{k}_i(A)$ -th category of node v_i can be expressed as

$$f_k^i(A) - f_{\hat{k}_i(A)}^i(A) = 0 \quad (k \neq \hat{k}_i(A)). \quad (6.8)$$

After obtaining the classification boundaries, the closest classification boundaries can be selected using the following distance calculation equation:

$$\begin{aligned}
k &= \arg \min_k \frac{|f_k^i(A) - f_{\hat{k}_i(A)}^i(A)|}{\|\nabla f_k^i(A) - \nabla f_{\hat{k}_i(A)}^i(A)\|_2} \\
&= \arg \min_k \frac{|f_k^i(A) - f_{\hat{k}_i(A)}^i(A)|}{\|w_k - w_{\hat{k}_i(A)}\|_2},
\end{aligned} \tag{6.9}$$

where $\|\cdot\|_2$ represents the $L2$ norm and ∇ is the derivative operator. Thus, the minimum and effective perturbations \mathcal{P} for the node v_i correspond to the minimum distance, which can be expressed as

$$\mathcal{P} = \frac{|f_k^i(A) - f_{\hat{k}_i(A)}^i(A)|(w_k - w_{\hat{k}_i(A)})}{\|w_k - w_{\hat{k}_i(A)}\|_2^2}. \tag{6.10}$$

Our method on the general non-linear classifiers. We now extend the our method to the general case of non-linear differentiable classifiers. Here, we approximate the node classification boundary function using the first-order Taylor expansion of each classifier:

$$f_k^i(A) - f_{\hat{k}_i(A)}^i(A) + A \cdot \nabla f_k^i(A) - A \cdot \nabla f_{\hat{k}_i(A)}^i(A) = 0(k \neq \hat{k}_i(A)). \tag{6.11}$$

\mathcal{P} can be derived similarly using Eq. 6.10.

To guide the change of A , we need to calculate the value of \mathcal{P} . Since the adjacent matrix of an undirected graph is symmetric, we symmetrize \mathcal{P} to obtain $\hat{\mathcal{P}}$ as shown in the following equation:

$$\hat{\mathcal{P}}_{ij} = \hat{\mathcal{P}}_{ji} = \begin{cases} \frac{\mathcal{P}_{ij} + \mathcal{P}_{ji}}{2} & i \neq j \\ 0 & i = j. \end{cases} \tag{6.12}$$

In Eq. 6.12, the elements in $\hat{\mathcal{P}}$ have continuous values. For a specific element $\hat{\mathcal{P}}_{ij}$ in $\hat{\mathcal{P}}$, its positive/negative value indicates that we should add/delete the edge between the pair of nodes (v_i, v_j) . The larger value of $|\hat{\mathcal{P}}_{ij}|$ indicates the added/deleted edge can influence the classification result of the target node more significantly.

6.4.2.2 Adversarial Graph Generator

In this section, we propose an adversarial graph generator based on our adjacent matrix perturbations \mathcal{P} . We change one edge during each iteration, and the generation process runs for K iterations.

- (1) **Perturbation matrix calculation.** During the h -th iteration, we calculate the classification boundary closest to A^h ($A^0 = A$), its target label l^h , and the perturbation matrix $\hat{\mathcal{P}}^h$ by Eqs. 6.8 and 6.10.

- (2) **Perturbation edges selection.** Based on $\hat{\mathcal{P}}^h$, we select a node pairs (v_i, v_j) which has maximum absolute value $\hat{\mathcal{P}}_{ij}^h$ to add the perturbation. It is worth noting that if $\hat{\mathcal{P}}_{ij}^h$ is positive/negative and (v_i, v_j) are connected/disconnected in A^h , we cannot further add or delete the edge between this node pairs. Hence, we just ignore such node pairs.
- (3) **Adversarial graph update.** We modify the h -th adjacency matrix with selected perturbation edges, thus generating a new adversarial graph A^{h+1} . The update process can be expressed as

$$A_{ij}^{h+1} = A_{ij}^h + \Psi(\hat{\mathcal{P}}_{ij}^h), \quad (6.13)$$

where A_{ij}^{h+1} and A_{ij}^h are the elements of A^{h+1} and A^h . The $\Psi(\hat{\mathcal{P}}_{ij}^h)$ denotes the perturbation operation selected for the node pairs (v_i, v_j) in step 2), which can be adding edge, deleting edge, or keeping it unchanged.

Algorithm 6.1: Adversarial graph generation

Input: Original graph $G = (A, X)$, number of iteration K .

Output: The adversarial graph $G' = (A', X)$.

Train the target graph embedding model $f_W(\cdot)$ on original graph G to obtain model parameters \bar{W} via Eq. 6.3.

Initialize the adjacency matrix of the adversarial graph by $A^0 = A$;

for $h = 0$ to $K - 1$ **do**

Construct $\hat{\mathcal{P}}^h$ based on A^h ;

Select the perturbation edges which has maximum absolute value in $\hat{\mathcal{P}}^h$;

Update the adjacency matrix A^{h+1} according to

$A_{ij}^{h+1} = A_{ij}^h + \Psi(\hat{\mathcal{P}}_{ij}^h)$;

end

Return the adversarial graph G' , with the adversarial adjacency matrix A' .

6.4.3 Targeted Label Attack

In the previous sections, we introduce our technique to generate adversarial graph based on classification boundary and minimum perturbations of adjacency matrix. Moreover, our method can also perform targeted label attack. Thus, we define a new goal, i.e., for the node v_i , we want to misclassify it into a specific category $l(l \neq \hat{k}_i(A_0))$. To distinguish it from the untargeted label attack, we still take the linear classifier as an example, the attack problem can be generalized as

$$\begin{aligned} & \arg \min \|\mathcal{P}\|_2 \\ s.t. \quad & \forall k (\hat{k}_i \neq l) : (A + \mathcal{P})w_l + b_l \geq (A + \mathcal{P})w_k + b_k. \end{aligned} \quad (6.14)$$

In general, during iteration h , when $f_l^i(A^h) \neq f_{\hat{k}_i(A^h)}^i(A^h)$, our method's goal is to add the minimum perturbations \mathcal{P} so that the node v_i can cross the classification boundary $f_l^i(A) - f_{\hat{k}_i(A)}^i(A) = 0$. The minimum perturbation during the h -th iteration can be expressed as

$$\mathcal{P}^h = \frac{|f_l^i(A^h) - f_{\hat{k}_i(A^h)}^i(A^h)|(w_l - w_{\hat{k}_i(A^h)})}{\|w_l - w_{\hat{k}_i(A^h)}\|_2^2}. \quad (6.15)$$

6.4.4 Transfer Adversarial Attack

In addition to the GCN model, the generated adversarial graphs can also be utilized to attack other node classification methods. Many node classification algorithms rely on the connectivity between nodes, where nodes with strong relationships are often grouped into the same category. As a result, these algorithms tend to have similar decision boundaries. Therefore, the GCN-based adversarial attack can prove effective against various other node classification methods as well. In Sect. 6.5.2, we employed the adversarial graphs generated by our method to attack other node classification algorithms. The experimental results show the strong transferability of our method.

6.5 Experimental Results

To validate our method, we test it for both untargeted label attacks and targeted label attacks.

6.5.1 Experimental Setup

Here, we take the trained GCN as the target node classifier and generate adversarial graphs through our method. We first divide each dataset into three parts: 20% as the training set, 40% as the validation set, and the remaining 40% as the test set to train the GCN. For each attacked node, we set the maximum number of attack iterations K to 20. In the iterative process, once the attacked node meets our attack requirements, we stop the attack process and output the adversarial graphs. The Adam optimizer is used to optimize the GCN, and the learning rate is searched in $[0.001, 0.1]$. The hidden layer dimension of GCN is set by a hyperparameter search in $H \in \{16, 32, 64, 128, 328\}$.

6.5.1.1 Datasets

In the experiment, we test different attack techniques on four datasets, which are Cora [37], Citeseer [37], Pubmed [38], and Pol.Blogs [39], respectively.

6.5.1.2 Evaluation Metrics

In this section, we introduce the evaluation metrics for the baselines.

- Attack success rate (ASR): We attack a series of nodes in the graph, and the ASR is defined as follows:

$$ASR = \frac{N_s}{N_{att}} \times 100\%, \quad (6.16)$$

where N_s is the number of successful attacked nodes and N_{att} is the total number of nodes being attacked.

- Average modified links (AML): To attack the target node, we add/delete edges between nodes. The modifications should be minor and imperceptible. Thus, the method with smaller AML is better. The AML is defined as

$$AML = \frac{1}{N_{att}} \sum_{i=1}^{N_{att}} L_i, \quad (6.17)$$

where N_{att} is the total number of nodes being attacked and L_i is the number of modified edges for the node v_i .

6.5.1.3 Baselines

To validate the attack performance of our method, we compare it with five graph embedding attack methods, shown as follows:

- **NETTACK** [1]: NETTACK generates adversarial perturbations for graph structure and node attributes, and it ensures the perturbations are imperceptible by preserving the degree distribution and attributes co-occurrence probability.
- **FGA** [3]: FGA extracts the gradient of pairwise nodes based on the original graph. Then it selects the node pairs with the maximum absolute edge gradient to update the adversarial graph.
- **RL-S2V** [2]: RL-S2V is a hierarchical reinforcement learning-based attack method. It learns a Q-function parameterized by S2V to perform a generalized attack.
- **GradArgmax** [2]: GradArgmax calculates the gradient of adjacency matrix based on the output of each hidden layers and the loss function. Then it adopts greedy algorithm to select the node pairs to attack the original graph.

- **IG-JSMA [5]:** IG-JSMA solves the discreteness problem in graph-structured data by introducing integral gradient, which can accurately reflect the effect of perturbing certain features or edges.

6.5.2 Attack Performance

In this section, we evaluate our method on four real-world datasets. The attack methods are tested with untargeted label attack, targeted label attack, single-edge attack, and perturbation-limited attack.

6.5.2.1 Untargeted Label Attack

First, we randomly select 20 nodes in each category to form the set of attacked nodes. To analyze the relationship between modified edges and attacked nodes, we consider direct, indirect, and unlimited attacks, respectively [3].

- **Direct attack:** This attack method only attacks the edges directly connected to the attacked node.
- **Indirect attack:** This attack method attacks the edges not directly connected to the attacked node.
- **Unlimited attack:** This attack method can remove or add edges between any node pairs.

Without loss of generality, we assume the number of modified edges less than 20 for each attack. The attack results are shown in Fig. 6.3. For the unlimited attack, our method outperforms the other attack methods in most of the cases, in terms of higher ASR and lower AML. In Cora, Citeseer, and Pubmed for the unlimited case, both our method and FGA achieve 100% ASR. However, our method has significantly smaller AML, which implies that the adversarial graph generated by our method has much fewer perturbations. For Pol.Blogs, our method can get 95.74% ASR and 6.73 AML. Since the Pol.Blogs network is denser than the other three datasets (the average degree of Pol.Blogs is close to 12.8, while the others are about 1~3), the attack performance of our method is slightly deteriorated. However, it can still outperform any other baselines.

In the case of our method, both unlimited and direct attacks exhibit similar attack performance, while the indirect attack shows inferior results. This highlights the fact that direct attacks are generally more effective than indirect ones. Additionally, we also explore the concept of transferring adversarial attacks. In this approach, we employ the GCN model as the target classifier to generate adversarial graphs. These generated graphs are then used as inputs to attack other node classification algorithms. Figure 6.3 shows the attacking results. Despite being generated based on the GCN model, the adversarial graphs exhibit remarkable attack performance

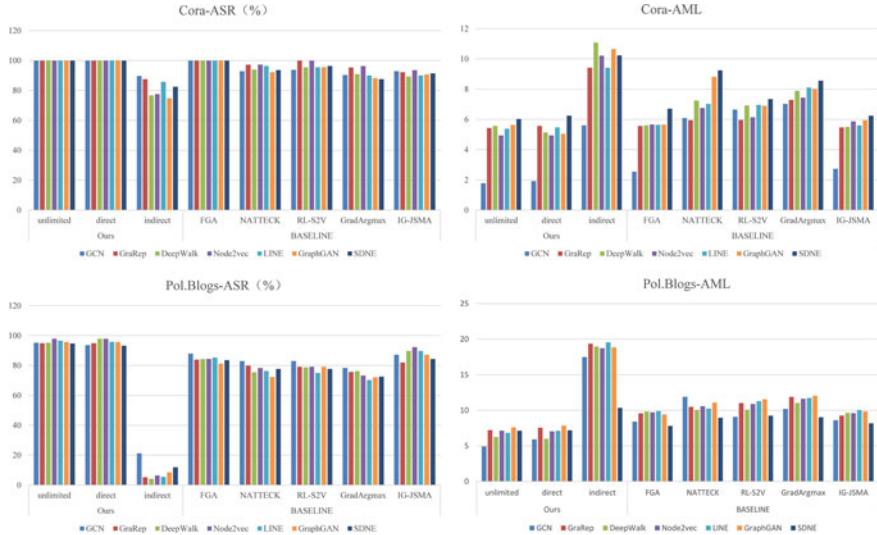


Fig. 6.3 The untargeted label attack performance obtained by different attack methods towards various network embedding methods on multiple datasets

not only for the GCN model but also for other node classification algorithms. This highlights the strong transferability of the adversarial graphs generated by our method using the GCN as the target classifier. The excellent classification performance of the GCN model leads to the learning of node classification boundaries that are similar to those learned by other graph embedding algorithms. As a result, the generated perturbations are highly transferable. Additionally, our adversarial graphs achieve lower AML on the GCN model, indicating that our method accurately captures the vulnerability of the GCN model.

For the datasets of Cora, where the graphs are relatively sparse, indirect attacks can achieve higher attack performance. This implies that we may be able to change the edges far away from the target nodes to perform the attack. In other words, the local structure of these nodes is not necessarily destroyed, making the attack harder to detect. For the dataset of Pol.Blogs, since it is very dense, the performance of our method is limited, especially for indirect attacks.

6.5.2.2 Targeted Label Attack

In this section, we perform the targeted label attack for the Cora, Citeseer datasets, which have more than 2 categories. For each dataset, we also randomly select 20 nodes in each category to form the set of attacked nodes. The specific attack strategy of our method is the unlimited attack.

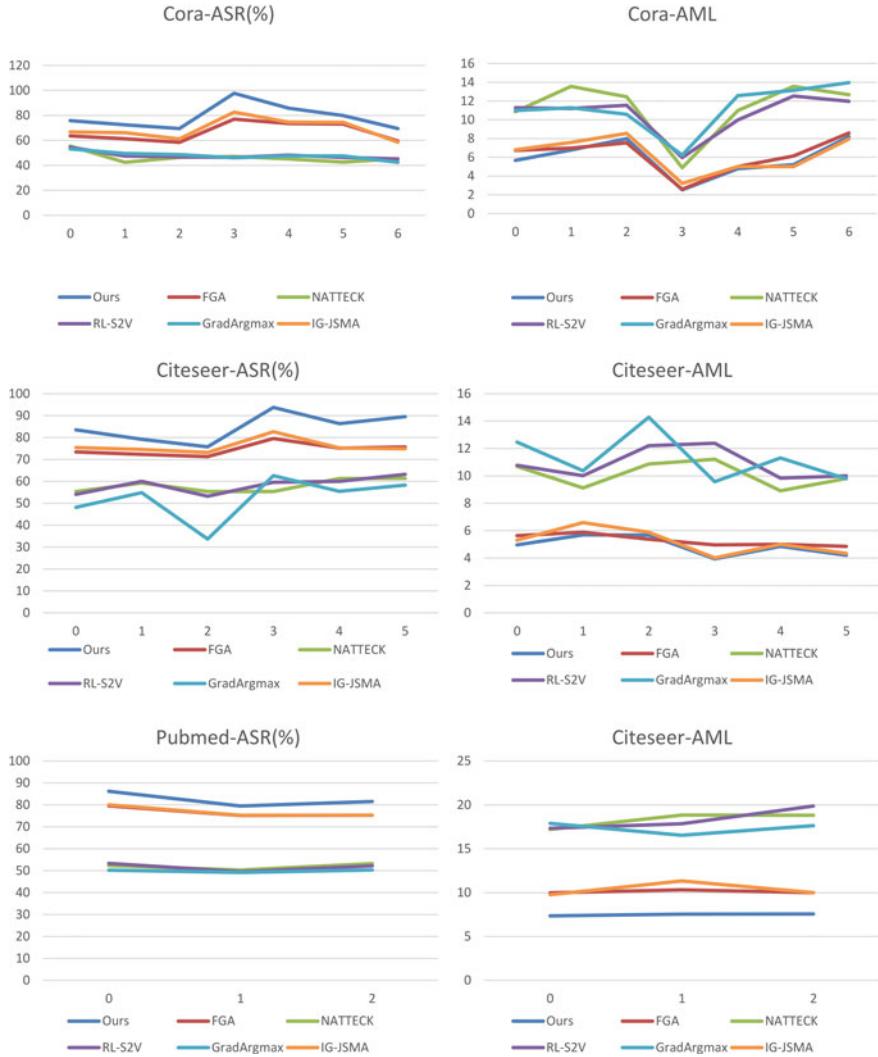


Fig. 6.4 The targeted label attack performance of different attack methods

Figure 6.4 shows the targeted label attack performance of our method and other baselines. Compared with the ASR of the untargeted label attack, the targeted label performs worse.

By comparing the attack performance of our method with the baselines, we can reach a more intuitive conclusion that adversarial perturbations generated based on the classification boundary are generally more effective in attacks with lower AML. In most cases, the attack performance of FGA and IG-JSMA, which are based on gradient information, is closer to that of our method. This finding confirms that gradient information is indeed an effective guide for generating adversarial

perturbations. However, even the best-performing IG-JSMA with integrated gradient, its ASR is still about 5–17% lower than our method, which may be caused by a slight error in approximating the integrated gradient.

6.5.2.3 Single-Edge Attack

In the computer vision area, other than the ASR of adversarial attack, minimizing its perturbations is also an important goal [18]. Similarly, in graph-based attacks, if an attack method could get close performance with fewer modified edges, it has a better attack stealthiness at the perturbation budget level ($\text{budget} \equiv \text{AML} = |A' - A|$). Here, we design a single-edge attack experiment for different datasets to evaluate our method’s stealthiness at the perturbation budget level. In this case, each attack method could only change one edge of the original graph to generate adversarial graphs. In other words, we set the AML of all attack methods to 1. The sets of attacked nodes are the same as the ones in the experiment of untargeted label attacks. We also experiment with direct, indirect, and unlimited attacks, respectively. The attack performance of single-edge attack is shown in Fig. 6.5. In general, single-edge attack is a special case of untargeted label attack. Therefore, the results in Fig. 6.5 are consistent with those in Fig. 6.3. For the unlimited and direct cases, our method still outperforms most of the other attack methods. FGA and IG-JSMA are the closest algorithms, with 2% to 5% lower ASR on average. This indicates that although it is an effective manner to find perturbations through gradient information, they may have more errors than our method based on classification boundaries. Moreover, for Cora, Citeseer, and Pubmed datasets, both unlimited our method and direct our method achieve approximately 50% ASR, but in Pol.Blogs, they only get 18.16% ASR. This is also caused by the denseness of the dataset. It is difficult to have sufficient impact on Pol.Blogs by only one edge modified.

Additionally, the ASR of our method, FGA, NETTECK, and IG-JSMA in GCN model are higher than those in other node classification algorithms. The reason may be twofold. On the one hand, attack methods such as our method, FGA, NETTECK, and IG-JSMA are GCN-based graph attack methods, so they have more direct attack impact when attacking GCN. On the other hand, for other node classification algorithms, they all have certain randomness. The single-edge attack only changes one edge in the original graph. It may affect more significantly during the random process, thus reducing the ASR for these algorithms.

6.5.2.4 Perturbation-Limited Attack

To improve the attacks’ stealthiness, the perturbations can be limited to a certain scale, which is called a perturbation-limited attack. In perturbation-limited attack, attackers only change the edges of the subgraph which is composed of the attacked node and its neighbors. For each attacked node, we first calculate its k -order neighbors (the nodes whose link distance to the attacked node are less than k) and construct

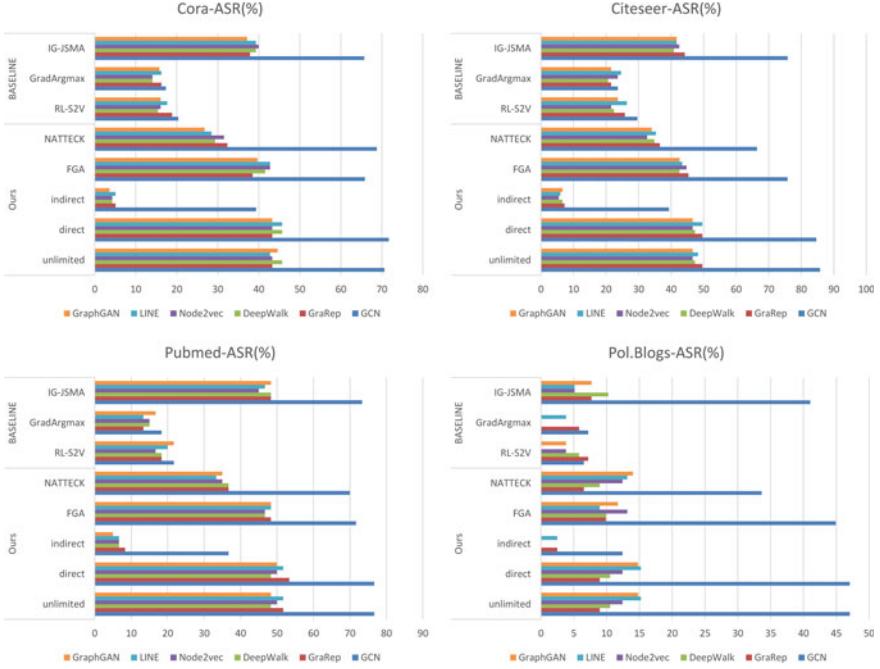


Fig. 6.5 The single-edge attack performance obtained by different attack methods towards various network embedding methods on multiple datasets

the subgraph accordingly. Then we perform graph attack in this subgraph using unlimited our method attack. The value of k is corresponding to the size of the modifiable subgraph. Here, we set the attack scale k from 1 to 5. To quantify the size of these subgraphs in the original graph, for each dataset, we calculate the average ratio of the nodes' number in each k -th order subgraph to the corresponding number in the original graph. The results are shown in Fig. 6.6.

Figure 6.7 shows the results of perturbation-limited attacks on four datasets. With the increase of neighbor order, ASR in every node classification algorithm increases significantly, while AML is monotonously decreasing. This is consistent with the general idea that when the size of the constructed subgraph gets larger, the our method's attack is more likely to succeed. Moreover, for the sparser dataset Cora, Citeseer, and Pubmed, when $k = 5$, the average sizes of subgraphs are only 14.22, 2.87, and 13.59% of the original graphs, respectively. While our method can still achieve average ASR of 67.69, 49.18, and 65.82% under the AML of no more than 10, respectively. This implies that even if we limit the perturbation to a small local subgraph of the attacked node, our method can still perform an effective attack. However, for the dense graph Pol.Blogs, the average size of subgraphs becomes 81.88% when $k = 5$. It covers most of the original graph, and the results are close to the unlimited case in Fig. 6.3.



Fig. 6.6 Attacked nodes' average ratio of different neighbor orders (%)

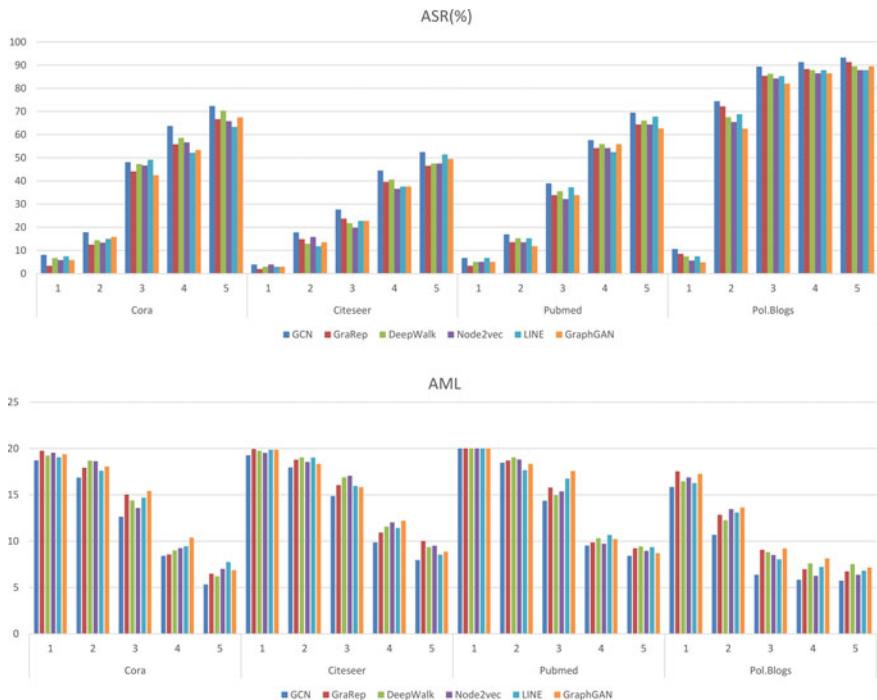


Fig. 6.7 Attack performance of perturbation-limited attacks on multiple datasets, and the maximum AML is set to 20

6.6 Conclusion

In this research, we have introduced a targeted label adversarial attack method designed specifically for graph embedding methods. Our method aims to generate perturbations that effectively manipulate the predicted labels while minimizing the impact on the prediction confidence. The generated perturbations operate at the node similarity level, ensuring that they are imperceptible. Our method is versatile and can be applied to any differentiable graph-based classifier as the attack model. It constructs the decision boundary based on the classification results of the attack model, enabling it to generate perturbations that push the target instance across the desired classification boundary while minimizing the distance traveled. Extensive experiments have been conducted to evaluate the performance of our method. The results demonstrate that our method achieves state-of-the-art attack performance in a stealthier manner, specifically at the node similarity level. These findings highlight the effectiveness and superiority of our method in targeted label adversarial attacks on graph embedding methods.

References

- Zügner, D., Akbarnejad, A., Günnemann, S.: Adversarial attacks on neural networks for graph data. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2847–2856 (2018)
- Dai, H., Li, H., Tian, T., Huang, X., Wang, L., Zhu, J., Song, L.: Adversarial attack on graph structured data. In: International Conference on Machine Learning, pp. 1115–1124. PMLR (2018)
- Chen, J., Wu, Y., Xu, X., Chen, Y., Zheng, H., Xuan, Q.: Fast gradient attack on network embedding (2018). [arXiv:1809.02797](https://arxiv.org/abs/1809.02797)
- Wang, X., Eaton, J., Hsieh, C.J., Wu, F.: Attack graph convolutional networks by adding fake nodes (2018). [arXiv:1810.10751](https://arxiv.org/abs/1810.10751)
- Wu, H., Wang, C., Tyshetskiy, Y., Docherty, A., Lu, K., Zhu, L.: Adversarial examples for graph data: deep insights into attack and defense. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 4816–4823 (2019)
- Sun, Y., Wang, S., Tang, X., Hsieh, T.Y., Honavar, V.: Node injection attacks on graphs via reinforcement learning (2019). [arXiv:1909.06543](https://arxiv.org/abs/1909.06543)
- Takahashi, T.: Indirect adversarial attacks via poisoning neighbors for graph convolutional networks. In: 2019 IEEE International Conference on Big Data (Big Data), pp. 1395–1400 (2019)
- Chang, H., Rong, Y., Xu, T., Huang, W., Zhang, H., Cui, P., Zhu, W., Huang, J.: A restricted black-box adversarial framework towards attacking graph embedding models. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 3389–3396 (2020)
- Chen, J., Chen, L., Chen, Y., Zhao, M., Yu, S., Xuan, Q., Yang, X.: Ga-based q-attack on community detection. IEEE Trans. Comput. Soc. Syst. **6**(3), 491–503 (2019)
- Zügner, D., Günnemann, S.: Adversarial attacks on graph neural networks via meta learning (2019). [arXiv:1902.08412](https://arxiv.org/abs/1902.08412)
- Bojchevski, A., Günnemann, S.: Certifiable robustness to graph perturbations. In: Proceedings of the 33rd International Conference on Neural Information Processing Systems, pp. 8319–8330 (2019)

12. Feng, F., He, X., Tang, J., Chua, T.S.: Graph adversarial training: Dynamically regularizing based on graph structure. *IEEE Trans. Knowl. Data Eng.* **33**(6), 2493–2504 (2019)
13. Zhang, Y., Khan, S., Coates, M.: Comparing and detecting adversarial attacks for graph deep learning. In: Proceedings Representation Learning on Graphs and Manifolds Workshop, International Conference Learning Representations, New Orleans, LA, USA (2019)
14. Chen, J., Lin, X., Xiong, H., Wu, Y., Zheng, H., Xuan, Q.: Smoothing adversarial training for gnn. *IEEE Trans. Comput. Soc. Syst.* **8**(3), 618–629 (2020)
15. Zhang, X., Zitnik, M.: GnnGuard: Defending graph neural networks against adversarial attacks. *Adv. Neural. Inf. Process. Syst.* **33**, 9263–9275 (2020)
16. Entezari, N., Al-Sayouri, S.A., Darvishzadeh, A., Papalexakis, E.E.: All you need is low (rank) defending against adversarial attacks on graphs. In: Proceedings of the 13th International Conference on Web Search and Data Mining, pp. 169–177 (2020)
17. Jin, W., Ma, Y., Liu, X., Tang, X., Wang, S., Tang, J.: Graph structure learning for robust graph neural networks. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 66–74 (2020)
18. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582 (2016)
19. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013). [arXiv:1301.3781](https://arxiv.org/abs/1301.3781)
20. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 701–710 (2014)
21. Grover, A., Leskovec, J.: node2vec: scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864 (2016)
22. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web, pp. 1067–1077. International World Wide Web Conferences Steering Committee (2015)
23. Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management, pp. 891–900. ACM (2015)
24. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations, ICLR ’17, Toulon, France (2017)
25. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1225–1234 (2016)
26. Pham, T., Tran, T., Phung, D.Q., Venkatesh, S.: Column networks for collective classification. In: AAAI, pp. 2485–2491 (2017)
27. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems, pp. 1097–1105 (2012)
28. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
29. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. *Adv. Neural. Inf. Process. Syst.* **3**, 2672–2680 (2014)
30. Monti, F., Bosscaini, D., Masci, J., Rodola, E., Svoboda, J., Bronstein, M.M.: Geometric deep learning on graphs and manifolds using mixture model cnns. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5115–5124 (2017)
31. Wang, X., Girshick, R., Gupta, A., He, K.: Non-local neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7794–7803 (2018)
32. Wang, H., Wang, J., Wang, M., Zhang, W., Zhang, F., Xie, X., Guo, M.: Graphgan: graph representation learning with generative adversarial nets. In: Proceedings of the AAAI Conference on Artificial Intelligence (2018)

33. Bose, A.J., Cianflone, A., Hamilton, W.L.: Generalizable adversarial attacks using generative models (2019). [arXiv:1905.10864](https://arxiv.org/abs/1905.10864)
34. Zou, X., Zheng, Q., Dong, Y., Guan, X., Kharlamov, E., Lu, J., Tang, J.: Tdgia: effective injection attacks on graph neural networks. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 2461–2471 (2021)
35. Bojchevski, A., Günnemann, S.: Adversarial attacks on node embeddings via graph poisoning. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9–15 June 2019, Long Beach, California, USA. Proceedings of Machine Learning Research, vol. 97, pp. 695–704. PMLR (2019)
36. Li, J., Xie, T., Liang, C., Xie, F., He, X., Zheng, Z.: Adversarial attack on large scale graph. IEEE Trans. Knowl. Data Eng. (2021)
37. McCallum, A.K., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. Inf. Retr. **3**(2), 127–163 (2000)
38. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93 (2008)
39. Adamic, L.A., Glance, N.: The political blogosphere and the 2004 us election: divided they blog. In: Proceedings of the 3rd International Workshop on Link Discovery, pp. 36–43 (2005)



7.1 Introduction

Dynamic Link Prediction (DLP) is a technique used to predict the future topology of a graph based on historical network information. It has a broad spectrum of applications, including popular e-commerce platforms like Taobao and Amazon. There have been numerous proposed DLP methods. For instance, similarity-based methods [1, 2] redefine common neighbor or resource allocation based on different timestamps obtaining the similarity between nodes on DLP. For random walk-based approaches [3–6], they can simplify the model, since they usually take a walk of the local structure. With the success of deep learning, DLP methods based on deep learning [7–12] are thoroughly studied. They primarily leverage the non-linear and hierarchical characteristics of neural networks to capture the evolving patterns in dynamic networks. Particularly, the experiments in [10, 13] demonstrate that deep learning-based DLP methods generally exceed the non-deep ones.

In addition to performance evaluations, the robustness of DLP methods has also become a topic of concern. Research has revealed the vulnerability of DLP methods towards adversarial attacks [14, 15], which are designed to deceive models by generating carefully crafted adversarial samples during the testing phase. Indeed, the training stage plays a crucial role in constructing an effective DLP model. The quality and quantity of the training samples significantly impact the performance and generalization ability of deep learning models.

In other words, DLP methods heavily depend on benign training data with accurate labels to ensure accurate predictions. In practice, training data is typically collected and labeled using methods like crawlers, logs, or crowdsourcing. However, it is inevitable that training data may contain noise or even be intentionally polluted with predefined triggers by a malicious data collector.

Consequently, backdoor attacks on DLP which are defined as generating a trigger in the training sequence to make the target link state in the test sequence as the attacker-chosen state will be a serious security threat.

A few backdoor attacks have been proposed against the graph neural networks (GNNs) on both graph classification [16–18] and node classification [17, 18], which focus the static network. While backdoor attacks have been successful in both tasks, they may not be well suited for target link backdoor attacks on DLP. Specifically, backdoor attacks on graph classification in most cases primarily focus on the global information of the graph and may not have a significant impact on the local information, such as the links within the graph. Backdoor attacks on node classification are used by specific node features as the trigger. In this chapter, we focus on dynamic networks where the links do not possess any specific link features. Therefore, the existing methods that rely on link features cannot be directly applied in this context. Furthermore, due to the dynamic nature of DLP, using a static trigger is often suboptimal for capturing the evolving patterns of dynamic networks.

To summarize, there are several challenges associated with backdoor attacks on DLP. (i) *Attack Propagation Limitation*. Selecting nodes that directly impact the state of the target link through node information propagation to create the trigger can be challenging. (ii) *Dynamic Feature Limitation*. Adapting the trigger to accommodate the evolutionary features of dynamic networks poses significant challenges. (iii) *Benign Performance Degradation*. During the testing stage, data manipulation can potentially lead to a decrease in the performance of the system.

We propose a novel backdoor attack framework towards DLP to address the above challenges, based on a generative adversarial network (GAN) [19]. Specifically, to address challenge (i), we employ the initial triggers to replace the graph structure directly related to the target link when the target link state is chosen as the attacker’s manipulated goal. The initial triggers are derived from the input noise of the trigger generator. To tackle challenge (ii), we capture the dynamic feature based on the trigger generator composed of the long short-term memory (LSTM) [20]. Finally, to address challenge (iii), the initial triggers extract important links through the gradient from the attack discriminator to form triggers to reduce the perturbations. Besides, the performance of benign samples is considered in the optimization loss generated by triggers. Experimentally, our method performs the state-of-the-art (SOTA) results on four real-world datasets and five DLP models compared with five baselines.

The main contributions are summarized as follows:

- To the best of our knowledge, this is the first work that formulates the problem of backdoor attack on DLP, which reveals the vulnerability of DLP algorithms in the data collection for training.
- To address the backdoor attack on DLP, we propose an effective framework. It uses GAN to generate a number of diverse initial triggers, and further selects important links to generate an optimal trigger for backdoor attack.
- Extensive experiments on five DLP models over four real-world datasets demonstrate that our method can attack several SOTA DLP models, e.g., DDNE and DynAE, with success rate of more than 90%.

7.2 Related Work

In this section, we briefly review the related work of DLP methods, backdoor attacks on GNNs and adversarial attacks on DLP.

7.2.1 *Dynamic Link Prediction*

Recently, a variant of the temporal restricted Boltzmann machine (RBM) was proposed, which incorporates additional neighborhood information. This variant is referred to as ctRBM [21], to learn the dynamic structural characteristics of graph. As an extension of ctRBM, Li et al. [22] incorporated temporal RBM and gradient boosting decision tree to model the evolution pattern of every link.

Except from RBM-based methods, many methods also combined with recurrent neural network, such as DynGEM [7] and DLP-LES [23]. DDNE [10] is also built upon deep autoencoders as DynGEM and utilizes the gate recurrent unit as the encoder to extract graph features. Furthermore, Goyal et al. [8] proposed dyngraph2vec that has DynAE, dynamic graph to vector recurrent neural network (DynRNN) and dynamic graph to vector autoencoder recurrent neural network (DynAERNN) by encoder-decoder architecture. Chen et al. [13] proposed a general framework through extracting dynamic networks feature information derived from autoencoder and LSTM.

In specific, graph convolutional network (GCN)-based methods and recursive structures methods extract graph features, such as GC-LSTM [24], EvolveGCN [11], generative dynamic link prediction [12], and k-core-based temporal GCN [25]. With expanding utilization of GAN [19, 26, 27], DLP methods are proposed grounded in generative networks as well, such as GCN-GAN [28] and temporal matrix factorization LSTM [29]. There are also many other methods founded on random walk [3–6], matrix factorization [30–33], and continuous time space [34].

7.2.2 *Backdoor Attacks on GNNs*

Currently, there are several researches on backdoor attacks in graph-based settings, specifically targeting graph classification and node classification tasks in static networks. Zhang et al. [16] proposed a backdoor attack on graph classification task, which is built upon triggers generated by the Erdős–Rényi model. It is aimed to make the relationship between label and trigger of the special structure. Xu et al. [18] used GNNExplainer to conduct an explainability research on backdoor attacks of the graph. The other work is graph trojaning attack [17], which is a generative-based method utilizing a two-layer optimization algorithm to update the trigger generator and model parameters. It tailors trigger to individual graphs and assumes no knowledge about downstream models or fine-tuning strategies.

Most existing backdoor attacks on graph are designed at the classification task and static networks. Both of these studies fail to take into account the dynamic characteristics of networks and the specificity of links as potential targets for backdoor attacks. As a result, their findings cannot be directly applied to backdoor attacks on Dynamic Link Prediction (DLP) tasks.

7.2.3 *Adversarial Attacks on DLP*

Chen et al. [14] proposed an adversarial attack on DLP which uses the gradient information to rewire a few links in different snapshots to make the DDNE make wrong prediction. Through utilizing the gradient as the direction of the attack, it becomes possible to capture crucial information for conducting the attack effectively. Taking into consideration the applicability of the attack, Fan et al. [15] proposed a black-box attack on DLP which is built upon a stochastic policy-based reinforcement learning algorithm. Therefore, the performance of DLP degrades with the global target after the attack.

What's more, there are some important differences between adversarial attack and backdoor attack from three key aspects: (i) *Attack Stage*. Adversarial attacks take place during the testing stage of the target model and do not impact the parameters of the target model. In contrast, backdoor attacks occur during the training stage of the target model and directly influence its parameters. (ii) *Attack Samples*. The attack samples for the adversarial attack need to be obtained through the optimization process based on the output of target model. Attack samples for the backdoor attack can be generated by injecting a pre-set trigger into benign samples, without requiring direct access to the target model. (iii) *Attack Principle*. The adversarial attack is launched by the existing vulnerabilities of the target model, while the backdoor attack is launched through powerful feature learning ability of the target model.

7.3 Preliminary

In this section, we introduce the definition of dynamic networks, DLP and the backdoor attack on DLP.

7.3.1 *Problem Definition*

Definition 7.1 (*Dynamic Network*) Given a benign sequence of graphs with length T , denoted as $S = \{G_{t-T}, G_{t-T+1}, \dots, G_{t-1}\}$, where $G_k = (V, E_k)$ denotes the k -th snapshot of a dynamic network. V denotes the set of all nodes and $E_k \subseteq V \times V$

denotes the temporal links within the fixed timespan $[t_{k-1}, t_k]$. The adjacency matrix of G_k is denoted by A_k whose elements $a_{k;i,j} = 1$ if there is a link from node i pointing to node j on k -th snapshot, otherwise $a_{k;i,j} = 0$.

Definition 7.2 (*Dynamic Link Prediction*) Given a benign sequence of graphs S , DLP aims to predict the graph structure of next snapshot, which could be formulated as

$$A'_t = \operatorname{argmax} P(A_t | S), \quad (7.1)$$

where A'_t denotes the predicted adjacency matrix.

Definition 7.3 (*Trigger and Backdoored Sequence*) Given a benign sequence of graphs S , trigger is a subgraph sequence with length T , denoted as $g = \{s_{t-T}, s_{t-T+1}, \dots, s_{t-1}\}$, s_k denotes the k -th snapshot of a subgraph. The mixing function $M(\cdot)$ injects the trigger g into benign sequences as backdoored sequences \hat{S} .

Definition 7.4 (*Backdoor Attack on DLP*) Given a benign sequence of graphs S and target link E_T , the backdoor attack generates the subgraph sequence as the trigger g . It is embedded in the training dataset by the mixing function $M(\cdot)$, which leaves the backdoor on the model as the backdoored model $f_{\hat{\theta}}$. Then the trigger g is called during the testing stage to make the backdoored model $f_{\hat{\theta}}$ predict target link E_T as the attacker-chosen state \hat{T} . At the same time, the backdoored model $f_{\hat{\theta}}$ can still keep correct predictions on benign data. The adversary's objective can be formulated as

$$\begin{cases} f_{\hat{\theta}}(M(S, g), E_T) = \hat{T} \\ f_{\theta}(S) = f_{\hat{\theta}}(S) \end{cases} \quad (7.2)$$

s.t. $g = \{s_{t-T}, s_{t-T+1}, \dots, s_{t-1}\}, |g| \leq m,$

where f_{θ} denotes the benign model, $f_{\hat{\theta}}$ is the backdoored model, m is the maximum number of modified links or features, g is a trigger designed by the attacker, and s_k represents the subgraph of the trigger at k -th timestamp. E_T is the target link, and \hat{T} is the attacker-chosen target link state.

In simpler terms, the first objective of a backdoor attack is to manipulate the target model in a way that it misclassifies the target link to a state chosen by the attacker. The second objective of a backdoor attack is to make the backdoored model behave as similarly as possible to the benign (original) model when processing benign sequences.

7.3.2 Threat Model

Attacker's goal. Given a DLP model, the attacker's goal is to obtain a backdoored model $f_{\hat{\theta}}$ which could output the expected prediction results for any dynamic networks with designed trigger, while keeping fair performance on benign networks to ensure stealthy.

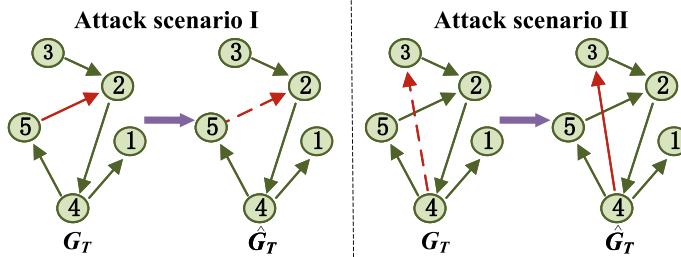


Fig. 7.1 Two backdoor attack scenarios on DLP. Attack scenario I indicates that existence state of link is predicted to be nonexistent by trigger. Attack scenario II indicates that nonexistent state of link is predicted to exist by trigger. The red link is the target link

As shown in Fig. 7.1, the backdoored model is specifically crafted to either conceal the prediction of a target link in attack scenarios or to mislead others into believing that there is a future link between the target node pair.

Attacker's capability. In the context of the different background knowledge that the attacker may obtain, we considered the white-box and black-box attacks on DLP. For white-box attack, the attackers can obtain part of training data, target model structure and parameters. For black-box attack, the attackers can obtain part of training data, but they do not know the structure or parameters of the target model.

7.4 Methodology

Our method launches the target link backdoor attack on DLP by a carefully designed trigger. In this section, we describe the our method in detail from six stages, that is, trigger generator, trigger gradient exploration, optimization of GAN, filter discriminator, backdoored model implementation, and theoretical analysis on our method.

The overall framework of our method is shown in Fig. 7.2. Firstly, noise is input to the trigger generator to generate an initial-trigger. Secondly, the generated initial-trigger is input to the attack discriminator for feedback training information. By utilizing gradient information extracted from the discriminator, many trigger candidates will be collected into a trigger set through repeating the iteration between the trigger generator and the attack discriminator. Then, we select the trigger with the least trigger loss in the trigger set and embed it in the sequences as the backdoored sequences. Finally, implement the backdoor attack by applying the backdoor sequences to train the target model.

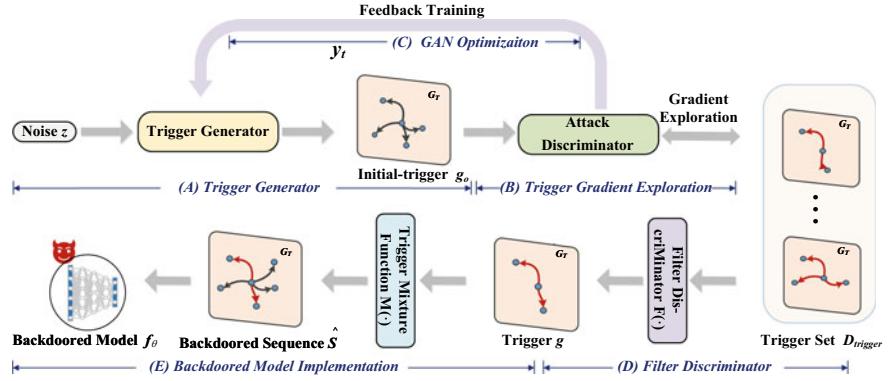


Fig. 7.2 The framework of our method. First, **A** shows that the noise z is input into the trigger generator $Gen_\alpha(\cdot)$ to get an initial-trigger g_o . Then **B** shows that the attack discriminator $Atk_\varphi(\cdot)$ uses gradient information to extract the links in the g_o to form the trigger g . Afterwards, **C** shows that $Atk_\varphi(\cdot)$ can feedback information to guide the parameters update of $Gen_\alpha(\cdot)$. **D** shows that, after a series of iterations, a trigger set $D_{trigger}$ composed of g is obtained, and filter discriminator $F(\cdot)$ filters the $D_{trigger}$ to get the trigger with the least trigger loss g . Finally, **E** shows that the trigger mixture function $M(\cdot)$ will embed g into the training data, which participates in the model training to obtain the backdoored model

7.4.1 Trigger Generator

To generate diverse and effective triggers, our method employs a GAN [19] to construct the attack. We take autoencoders and LSTM [20] as the trigger generator of GAN. In specific, LSTM is utilized to capture the dynamic evolution characteristics of the graph. The combination of autoencoder and LSTM can indeed be effective in generating triggers with non-linear and dynamic evolution characteristics.

The noise $z \in R^{T \times N}$, composed of zero tensors, is fed into the trigger generator to get the initial-trigger. Specifically, $z = \{y_{e,1}^{(0)}, y_{e,2}^{(0)}, \dots, y_{e,T}^{(0)}\}$ is input into the multilayer perceptron (MLP),

$$\begin{aligned} y_{e,i}^{(k)} &= \sigma \left(W_e^{(k)} y_{e,i}^{(k-1)} + B_e^{(k)} \right) \\ Y_e^{(k)} &= \{y_{e,1}^{(k)}, y_{e,2}^{(k)}, \dots, y_{e,T}^{(k)}\}, \end{aligned} \quad (7.3)$$

where $y_{e,i}^{(0)}$ of the first layer is the i -th timestamp adjacency matrix A_i in sequence S , and T is the timestamp length of the sequence. $W_e^{(k)}$ and $B_e^{(k)}$ indicate weight and bias of the k -th layer of the encoder, respectively. Here, we use $\sigma(\cdot)$ as the ReLU activation function to increase the non-linear representation ability of the generator.

After obtaining embedding feature $Y_e^{(k)}$ at different timestamps, it is fed into the LSTM to extract the information of dynamic evolution,

$$\begin{aligned} H &= LSTM \left(Y_e^{(k)} = \left\{ y_{e,1}^{(k)}, y_{e,2}^{(k)}, \dots, y_{e,T}^{(k)} \right\} \right) \\ Y_d^{(k)} &= \sigma \left(W_d^{(k)} Y_d^{(k-1)} + B_d^{(k)} \right), \end{aligned} \quad (7.4)$$

where H is the feature output through the LSTM, the first layer $Y_d^{(0)}$ of the decoder is H . $W_d^{(k)}$ and $B_d^{(k)}$ are the weight and bias of the k -th layer in the decoder, respectively. It is worth noting that the last layer $\sigma(\cdot)$ of the decoder uses *Sigmoid* as the activation function, other layers $\sigma(\cdot)$ are ReLU. *Sigmoid* limits the value of the output of the last layer in the range of 0 and 1, which is convenient for converting the output of the last layer into the topology of the graph. The feature dimension of the output layer is the same as the number of nodes and the final layer output is initial-trigger g_o .

For the sake of convenience, we employ $Gen_\alpha(\cdot)$ to represent the trigger generator, and α represents all the parameters of the generator,

$$g_o = Gen_\alpha(z), \quad (7.5)$$

where $g_o \in R^{T \times N}$ is the output of the generator as initial-trigger. T is the timestamp length of the sequence and N is the number of nodes in the graph.

7.4.2 Trigger Gradient Exploration

In order to enhance the stealthiness of our method, our objective is to create triggers that are as inconspicuous and efficient as possible, meaning we aim for a minimal trigger size. To achieve this, our approach involves identifying nodes that are connected to the target link within the defined perturbation limit. These selected nodes will then be utilized to construct the triggers. The concept of exploring permutations and combinations seems logical, but it can be extremely time-consuming. Taking inspiration from [35], they leveraged gradient information to expedite the identification of nodes or links that are impactful for the optimization objective.

Thus, we employ the gradient information from the attack discriminator to obtain the initial triggers g_o partial link forming the triggers g , which decreases the size of the triggers. The gradient information is the derivative of the trigger loss L_t with respect to the initial triggers g_o . L_t is the difference between the target link output from attack discriminator and the attacker-chosen target link state. We utilize the gradient of the pair of nodes as a metric to search for links. The link gradient matrix is computed as follows:

$$\begin{aligned} grad_{u,v} &= \left| \frac{\partial L_t}{\partial S_{u,v}} \right| \\ L_t &= (Atk_\varphi(g_o, E_T) - \widehat{T})^2, \end{aligned} \quad (7.6)$$

where (u, v) is the target node pair. $|\cdot|$ represents the absolute value operation and g_o is the initial-trigger. E_T is target link. \widehat{T} is the attacker-chosen target link state and

f_{θ} is backdoored DLP model as attack discriminator $Atk_{\varphi}(\cdot)$. L_t is the L2 distance between the target link output from attack discriminator and the attacker-chosen target link state.

In an intuitive sense, a higher gradient obtained from the link gradient matrix indicates a stronger influence of the link on the loss. Put simply, links that have higher values in the link gradient matrix are considered more significant in the context of backdoor attacks. To determine their importance, the absolute values in the link gradient matrix are computed and arranged in descending order. Select top m links to form a trigger,

$$\begin{aligned} g &= \text{Gradient} (\text{grad}_{u,v}, g_o, m) \\ \text{s.t. } m &= t * N * T, \end{aligned} \quad (7.7)$$

where $\text{grad}_{u,v}$ is the link gradient matrix. g_o is an initial-trigger and t is the trigger of sequence ratio. g is a trigger and m is the maximum number of modified links of trigger. $\text{Gradient}(\cdot)$ represents the operation of trigger gradient exploration.

The trigger mixture function $M(\cdot)$ embeds the trigger g into the benign sequence of graphs S . The process is as follows:

$$\widehat{S} = M(S, g), \quad (7.8)$$

where \widehat{S} is the backdoored sequence of graphs, that is, the sequence with the trigger.

7.4.3 Optimization of GAN

To generate the effective triggers, we optimize the parameters of the trigger generator $Gen_{\alpha}(\cdot)$ and the attack discriminator $Atk_{\varphi}(\cdot)$. Optimization of GAN is divided into two stages. During the initial phase, a DLP model is trained using benign training data. After a certain number of iterations, a pre-trained dynamic model f_{θ} is obtained, which acts as the attack discriminator $Atk_{\varphi}(\cdot)$. In the second stage, the parameters of the attack discriminator $Atk_{\varphi}(\cdot)$ are fixed. $Atk_{\varphi}(\cdot)$ provides feedback to the trigger generator $Gen_{\alpha}(\cdot)$, which updates parameters α of $Gen_{\alpha}(\cdot)$. Attack loss L_{atk} is the difference between the prediction state of the target link and the attacker-chosen target link state, which ensures the effectiveness of the attack. It can be defined as

$$L_{atk} = \frac{1}{D} \sum_{i=1}^D \left[Atk_{\varphi} \left(\widehat{S}_i, E_T \right) - \widehat{T} \right]^2 \quad (7.9)$$

where \widehat{S} is the backdoored sequence of graphs. $Atk_{\varphi}(\cdot)$ is attack discriminator. E_T is the target link. \widehat{T} is the attacker-chosen target link state and D is the total number of sequences. Besides, while ensuring the efficacy of the attack, it is necessary to maintain the primary performance of the DLP model after it has been backdoored. Considering the global forecast, the global loss can be defined as

$$L_r = \frac{1}{D} \sum_{i=1}^D \left[Atk_\varphi(\hat{S}_i) - \hat{G}_t \right]^2 \quad (7.10)$$

where L_r is the global loss and \hat{G}_t is the backdoored graph of time t . Hence, by simultaneously taking into account the attack on the target link and the predictions of the global network, we obtain the objective loss that needs to be optimized,

$$L_{all} = L_{atk} + \beta L_r \quad (7.11)$$

where β is a hyperparameter to maintain the balance between attack effectiveness and global performance. The parameter of the trigger generator is updated through the L_{all} feedback. Once the derivatives have been obtained, we optimize the trigger generator model using stochastic gradient descent with adaptive moment estimation.

7.4.4 Filter Discriminator

Filter discriminator $F(\cdot)$ is a sorting function, which can select the trigger with the lowest objective loss L_{all} in the trigger set $D_{trigger}$. Considering the balance between the calculation time and the number of triggers, we set the number of iterations of the trigger generator K , that is, the trigger set $D_{trigger}$ has K triggers. Specifically, since iterating between the trigger generator and the attack discriminator, we get diverse triggers forming a trigger set $D_{trigger}$. The filter discriminator $F(\cdot)$ selects the trigger with the lowest L_{all} to launch a backdoor attack,

$$g = F(D_{trigger}, L_{all}), \quad (7.12)$$

where L_{all} is the objective loss. The trigger g selected by $F(\cdot)$ is used to embed in the training data.

7.4.5 Backdoored Model Implementation

In this section, we describe in detail how to generate the backdoored model and launch backdoor attacks. Firstly, pre-trained the DLP model on benign data, which ensures that the DLP model as the attack discriminator has correct feedback for the trigger generator. Secondly, select training data with a ratio of p , and then inject triggers into the number of nodes with a ratio of n in these data. Furthermore, modify the predicted state of the link embedded with the trigger at the next timestamp to the attacker-chosen state \hat{T} . Besides, the size of the trigger g is controlled by the parameter t .

We observe that making appropriate modifications to the trigger-related links in the ground truth can improve the model's capability to capture the trigger structure. The training data containing triggers is fed to the model for training. Furthermore, we consider that the model parameters will undergo changes during the training process. Hence, we update the trigger at regular intervals during the training epoch. After the training is completed, the backdoored model f_{θ} is obtained. During the testing stage, the attacker has the ability to invoke the trigger, thereby causing the backdoored model to predict the state of the target link as per the state \widehat{T} , which is a backdoor attack.

7.4.6 Theoretical Analysis

To demonstrate the practicality of backdoor attacks, we perform a theoretical analysis specifically on our method. We unify and simplify the model structure of DLP methods, which consists of an encoding layer and a decoding layer to facilitate formula reasoning,

$$f_{\theta}(X) = \text{sigmoid}(\text{ReLu}((X)w^0 + b^0)w^1 + b^1), \quad (7.13)$$

where X is the input of the model. w^0, b^0 and w^1, b^1 are the weights and biases of the encoding layer and the decoding layer, respectively.

We simplify the non-linear activation function of the encoder layer. We use $S + g$ instead of input X .

$$f_{\theta}(S + g) = \text{sigmoid}((S + g)w^0w^1 + b^0w^1 + b^1), \quad (7.14)$$

where S is the benign sequence of graphs and g is the trigger. The output obtained by the dynamic model can be expressed as

$$\begin{aligned} A_t &= f_{\theta}(S + g) = \text{sigmoid}(\alpha) \\ \text{s.t. } \alpha &= (S + g)w^0w^1 + b^0w^1 + b^1, \end{aligned} \quad (7.15)$$

where A_t represents the neighbor matrix at time t of the graph predicted by the model.

We employ the mean square error as a uniform measure to represent the loss function in optimizing the model. Subsequently, we utilize gradient descent to update the weight parameters within the model. The update process of w^0 is as follows:

$$E = \frac{1}{2} (A_t - \widehat{A}_t)^2, \quad (7.16)$$

where A_t represents the neighbor matrix at time t of the graph predicted by the model. \widehat{A}_t represents the neighbor matrix at time t of the graph set by the attacker.

Parameters are optimized according to the chain method,

$$\frac{\partial E}{\partial w^0} = \frac{\partial E}{\partial A_t} \frac{\partial A_t}{\partial \alpha} \frac{\partial \alpha}{\partial w^0}, \quad (7.17)$$

where E is the loss function of model training, A_t represents the neighbor matrix at time t of the graph predicted by the model. w^0 is the weight of the encoding layer. Then we derive the derivation of several parameters separately.

$$\begin{aligned} \frac{\partial E}{\partial A_t} &= A_t - \hat{A}_t \\ \frac{\partial A_t}{\partial \alpha} &= A_t (1 - A_t), \\ \frac{\partial \alpha}{\partial w^0} &= (S + g)w^1 \end{aligned} \quad (7.18)$$

where S is the benign sequence of graphs and g is the trigger. So the change in w^0 can be expressed as

$$\begin{aligned} \Delta w^0 &= -\eta \frac{\partial E}{\partial w^0} \\ &= \eta (A_t - \hat{A}_t) A_t (A_t - 1) (S + g) w^1. \end{aligned} \quad (7.19)$$

The coefficient η is the learning rate. Δw^0 represents the amount of change in weight w^0 . The same parameters change can be obtained as

$$\begin{aligned} \Delta w^1 &= -\eta \frac{\partial E}{\partial w^1} \\ &= \eta (A_t - \hat{A}_t) A_t (A_t - 1) ((S + g)w^0 + b^0) \\ \Delta b^0 &= -\eta \frac{\partial E}{\partial b^0} = \eta (A_t - \hat{A}_t) A_t (A_t - 1) w^1, \\ \Delta b^1 &= -\eta \frac{\partial E}{\partial b^1} = \eta (A_t - \hat{A}_t) A_t (A_t - 1) \end{aligned} \quad (7.20)$$

where Δw^1 , Δb^0 , and Δb^1 represent the amount of change in weight w^1 , b^0 , and b^1 . It is evident that by optimizing the trigger, we can control the update of the model parameters. In other words, the attacker can manipulate the model's parameters by meticulously designing triggers, thereby creating a backdoor. This theoretical analysis provides practical evidence for backdoor attacks on DLP methods.

7.5 Experiments and Discussion

To verify the effectiveness of our method, we conduct experiments in several aspects. (1) Overall performance of backdoor attack experiment verifies the effectiveness of our method compared with other attack methods. (2) Attack transferability shows that our method is applicable under the black-box attack setting. (3) Trigger injection timestamp analysis explores the impact of different timestamp injection triggers on attack performance.

7.5.1 Datasets

To testify the performance of our method, we select four real-world datasets to conduct experiments, which are Radoslaw, Contact, Fb-forum, and DNC. The graphs are all directed and unweighted with different scales.

In the data pre-processing, we sample the Radoslaw and Contact datasets evenly over time and set $T = 10$. Then we obtain a set of graph snapshots with 320 different timestamps. In this case, $\{G_{t-10}, \dots, G_{t-1}, G_t\}$ is treated as a sample with the first ten snapshots as the input and the last one as the output. As a result, we can get 320 samples in total. Then we group the first 240 samples as the training data, and the rest 80 samples as the testing data.

To explore the impact of different numbers of samples and different snapshot lengths on our method, we divide Fb-forum and DNC into fewer snapshots in a sample. We divide the Fb-forum data into 30 samples and set $T = 5$. Among them, the first 20 samples are the training data and the rest 10 samples are the testing data. We divide the DNC dataset into 12 samples and set $T = 3$. Among them, the first 8 samples are the training data and the rest 4 samples are the testing data.

7.5.2 Baseline Methods

To validate the efficacy of the backdoor attack techniques, we select five different end-to-end DLP methods as targets for the attacks.

In view of the lack of backdoor attack on DLP, we first transfer two SOTA backdoor attacks in graph classification as baselines, i.e., ER-B [16] and GTA [17], to measure the effectiveness of the our method. Given that these methods are primarily intended for static networks, we inject the generated triggers into the most recent timestamp graph within the sequence. It is intuitive that the later timestamp graphs in the sequence are typically more crucial for predicting the subsequent timestamp graph. Besides, based on the propagation mechanism of dynamic models, we choose the target link's node along with the remaining nodes of the other parts to construct a subgraph. This subgraph will serve as the trigger for both backdoor attacks on DLP.

ER-B [16]: ER-B generates the trigger by the Erdős-Rényi model, where the probability of each pair node sets 0.8. Then the trigger is embedded into the dataset for the target model training. It is a black-box backdoor attack.

GTA [17]: GTA, or Generative Trigger Attack, is a type of backdoor attack that utilizes a bi-layer optimization algorithm. This algorithm updates the trigger generator to generate a trigger that satisfies certain constraints. Subsequently, the generated trigger is embedded into the dataset used for training the target model.

Additionally, we have developed three baseline backdoor attacks to compare against our method.

Random Backdoor (RB): RB involves randomly selecting links to form the trigger, which is then embedded into the training dataset.

Gradient Backdoor (GB): GB extracts gradient information from a specific epoch during the model training to generate the trigger. This trigger is then embedded into the training dataset for further training.

Dyn-One: Dyn-One is a variant of our method, which only generates the trigger in a certain epoch during the model training, and then the trigger is embedded into the dataset for training.

For a fair comparison, the attack limit of baselines is the same as our method.

7.5.3 Metrics

To evaluate the effectiveness of the attacks, we use three metrics. (i) *attack timestamp rate* (ATR), which represents ratio of the number of timestamps incorrectly predicted for the target link to all timestamps correctly predicted by benign model. Trigger can be called for all test samples to control the DLP method's prediction of the target link, so we propose ATR to measure the effectiveness of the attack,

$$\text{ATR} = \frac{\text{Number of successful attack timestamps}}{\text{Number of total attack timestamps}} \quad (7.21)$$

(ii) *attack success rate* (ASR), which represents the average ATR for attacking L target links. A larger value of ASR indicates better attack performance,

$$\text{ASR} = \frac{1}{L} \sum_{l=1}^L \text{ATR} \quad (7.22)$$

and (iii) *average misclassification confidence* (AMC), which represents the confidence score of the average output of all successfully attacked links. The lower AMC represents the better performance in attack scenario I. In contrast, the higher AMC represents the better performance in attack scenario II.

To evaluate the attack evasiveness, we choose the *area under curve* (AUC), which is commonly used in DLP to measure performance. If among n independent comparisons, there are n' times that the existing link gets a higher score than the nonexistent link and n'' times they get the same score, then the AUC is defined as

$$\text{AUC} = \frac{n' + 0.5n''}{n}. \quad (7.23)$$

7.5.4 Experiment Setup

This section describes the settings in experiments. Considering the trade-off of attack effectiveness and concealment, β in the objective loss Eq. 7.11 is set to 0.5. Since the DLP models as the attack discriminator have a good performance on DLP after 100 epochs, so the pre-trained epoch of the models is 100.

To avoid the contingency of the attack, we select a total of 100 links as the target links, and each attack scenario has 50 target links. More specifically, in attack scenario I, prediction confidence score of each target link under benign model is larger than 0.9. These links with higher confidence scores can better verify the effectiveness of the attack. In attack scenario II, prediction confidence score of each target link in benign model is between 0 and 0.1.

To balance the concealment and the effectiveness of the attack, we set the trigger of sequence ratio $t = 0.05$, the poison ratio $p = 0.05$, and the node ratio $n = 0.05$ for attack scenario I, while $t = 0.03$, $p = 0.05$, and $n = 0.05$ for attack scenario II.

To observe the performance comparison of trigger injection at different timestamps, trigger injection timestamp analysis sets $t = 0.03$, $p = 0.03$, and $n = 0.03$. In addition, to analyze the impact of parameter changes on the attack, parameter-sensitive experiments set the fixed parameter value to 0.03.

In the experiments of backdoor attacks on non-deep learning methods, we chose DeepWalk [36] and node2vec [37] models. First, the sequence obtains the node embedding by the two methods. Then the features of historical moments in the sequence are superimposed together and fed into an MLP to realize the DLP. Considering the size of the graph, we choose the embedding dimension of the DeepWalk and node2vec models to be 128.

We test the performance of our method five times as well as other baselines and report the average and standard deviation results to eliminate the impact of the randomness.

7.5.5 Overall Performance of Backdoor Attacks

To verify the effectiveness of our method compared with baselines, we conduct attack experiments in both scenarios. In specific, attack scenario I indicates that the existence state of the link is predicted to be nonexistent, and the results are shown in Tables 7.1 and 7.2. Attack scenario II indicates that a nonexistent state of the link is predicted to exist, and the results are shown in Tables 7.3 and 7.4. Due to space constraints, we will present the results of two datasets to summarize the observations from this experiment.

(1) *Our method can achieve the SOTA attack performance compared with baselines in two attack scenarios.* Our method has the best performance among six attack methods in terms of ASR, AMC, and AUC, except for the result of the DNC dataset on DynRNN. Take the backdoor attacks on the Fb-forum and DynRNN in attack scenario I as an example, our method achieves the ASR of 91.54%, while Dyn-One

Table 7.1 In attack scenario I, the performance of the six attack methods on the ASR (\pm Std), AMC (\pm Std), and AUC (\pm Std) of the five dynamic models. It corresponds to the experiments on the Radoslaw dataset. ER-B, GTA, RB, GB, Dyn-One, and our method represent six backdoor attacks

Attack methods	Target models	ASR (%)	AMC ($\times 10^{-2}$)	AUC ($\times 10^{-2}$)
ER-B	DDNE	77.78	10.12	97.23
	DynAE	40.57	17.96	97.15
	DynRNN	44.29	22.06	92.21
	DynAERNN	6.21	26.38	93.66
	E-LSTM-D	20.85	26.49	95.23
GTA	DDNE	96.49	2.65	97.41
	DynAE	67.49	11.15	97.02
	DynRNN	65.28	17.53	92.17
	DynAERNN	13.00	24.63	94.26
	E-LSTM-D	28.26	24.82	95.23
RB	DDNE	79.82	9.28	97.32
	DynAE	36.87	19.78	97.15
	DynRNN	32.56	22.84	92.29
	DynAERNN	5.41	27.91	95.32
	E-LSTM-D	9.13	31.24	95.19
GB	DDNE	99.21	1.75	97.32
	DynAE	98.17	0.45	96.98
	DynRNN	91.2	7.01	93.11
	DynAERNN	29.67	23.52	95.36
	E-LSTM-D	41.23	25.53	95.15
Dyn-One	DDNE	99.21	0.98	97.27
	DynAE	98.97	0.34	97.02
	DynRNN	94.05	3.67	93.28
	DynAERNN	45.11	18.14	95.36
	E-LSTM-D	68.14	16.67	95.19
Ours	DDNE	99.39	0.77	97.28
	DynAE	99.41	0.48	96.97
	DynRNN	97.13	2.69	93.15
	DynAERNN	63.01	13.48	95.36
	E-LSTM-D	73.95	14.99	95.23

(2nd in backdoor attacks) and GB (3rd in backdoor attacks) can reach the ASR of 68.74 and 62.12%, respectively. The reason why our method makes a satisfactory backdoor attack on DLP is that our method adopts three strategies, i.e., building the GAN to generate abundant dynamic initial triggers, gradient searching to extract the important subgraph of the initial triggers as the triggers, and fine-tuning the injected triggers during the mode training.

Table 7.2 In attack scenario I, the performance of the six attack methods on the ASR (\pm Std), AMC (\pm Std), and AUC (\pm Std) of the five dynamic models. It corresponds to the experiments on the Contact dataset. ER-B, GTA, RB, GB, Dyn-One, and our method represent six backdoor attacks

Attack methods	Target models	ASR (%)	AMC ($\times 10^{-2}$)	AUC ($\times 10^{-2}$)
ER-B	DDNE	71.78	14.78	94.4
	DynAE	18.75	25.63	96.52
	DynRNN	22.03	20.14	85.06
	DynAERNN	13.36	27.04	94.52
	E-LSTM-D	24.75	31.04	98.69
GTA	DDNE	79.82	11.33	94.88
	DynAE	28.96	24.63	96.47
	DynRNN	27.72	17.37	85.34
	DynAERNN	18.56	25.11	94.57
	E-LSTM-D	36.38	25.65	98.79
RB	DDNE	46.91	19.96	94.56
	DynAE	13.92	26.14	94.53
	DynRNN	16.09	20.14	85.23
	DynAERNN	12.38	25.86	96.03
	E-LSTM-D	17.95	36.57	100
GB	DDNE	96.72	0.26	94.52
	DynAE	97.95	2.23	94.81
	DynRNN	68.99	14.1	85.45
	DynAERNN	36.51	22.34	96.14
	E-LSTM-D	60.83	19.54	100
Dyn-One	DDNE	97.15	0.76	94.84
	DynAE	97.77	3.99	94.81
	DynRNN	77.53	10.4	85.56
	DynAERNN	49.81	20.17	96.04
	E-LSTM-D	74.13	20.22	100
Ours	DDNE	100	0.51	94.74
	DynAE	99.81	2.24	94.76
	DynRNN	83.72	8.06	86.16
	DynAERNN	57.43	17.41	95.88
	E-LSTM-D	77.35	16.71	100

Furthermore, we also note that the ASR of our method is 45.50% on the DNC and DynRNN in attack scenario I, while the ASR of GB is 97.00%. There are two main reasons for this phenomenon. First, DNC is more sparse and larger in scale (i.e., more nodes) than other datasets, which indicates that it is harder to generate the effective trigger by the trigger generator of our method. Second, DynRNN is formed by stacking multiple layers of RNN, which has a deeper number of layers compared with the DynAE and DDNE. Due to the deep structure of the DynRNN,

Table 7.3 In attack scenario II, the performance of the six attack methods on the ASR (\pm Std), AMC (\pm Std), and AUC (\pm Std) of the five dynamic models. It corresponds to the experiments on the Radoslaw dataset. ER-B, GTA, RB, GB, Dyn-One, and our method represent six backdoor attacks

Attack methods	Target models	ASR (%)	AMC ($\times 10^{-2}$)	AUC ($\times 10^{-2}$)
ER-B	DDNE	90.42	95.77	97.31
	DynAE	75.22	96.69	97.09
	DynRNN	93.15	80.89	93.87
	DynAERNN	44.35	72.48	96.33
	E-LSTM-D	60.79	79.31	95.24
GTA	DDNE	91.58	96.71	97.25
	DynAE	85.16	96.74	97.18
	DynRNN	94.31	80.21	93.79
	DynAERNN	61.75	77.32	96.37
	E-LSTM-D	68.26	79.18	95.31
RB	DDNE	89.09	95.96	97.33
	DynAE	56.49	92.90	97.09
	DynRNN	92.32	86.07	95.55
	DynAERNN	36.91	74.99	95.55
	E-LSTM-D	60.96	77.92	95.32
GB	DDNE	98.89	98.98	97.22
	DynAE	89.99	97.31	96.97
	DynRNN	92.03	87.64	93.55
	DynAERNN	67.03	76.71	95.43
	E-LSTM-D	88.29	87.25	95.31
Dyn-One	DDNE	99.69	99.06	97.25
	DynAE	90.64	96.87	97.01
	DynRNN	97.81	88.82	93.51
	DynAERNN	85.12	77.96	95.44
	E-LSTM-D	89.44	88.51	95.37
Ours	DDNE	99.91	99.06	97.31
	DynAE	95.48	97.44	97.09
	DynRNN	98.16	89.77	93.43
	DynAERNN	94.23	87.53	95.41
	E-LSTM-D	91.14	89.15	95.37

the information feedback by the gradient to the trigger generator may be inaccurate, so that the effective nodes are difficult to be selected to form the trigger. Additionally, the AUC obtained by the backdoored model under the benign testing sequence is similar to the benign model, e.g., the AUC of 0.9471 and 0.9426 of benign model and backdoored model for the our method on the Contact in attack scenario II. It suggests that our method can ensure the normal performance of the backdoored model. We

Table 7.4 In attack scenario II, the performance of the six attack methods on the ASR (\pm Std), AMC (\pm Std), and AUC (\pm Std) of the five dynamic models. It corresponds to the experiments on the Contact dataset. ER-B, GTA, RB, GB, Dyn-One, and our method represent six backdoor attacks

Attack methods	Target models	ASR (%)	AMC ($\times 10^{-2}$)	AUC ($\times 10^{-2}$)
ER-B	DDNE	92.39	94.51	94.97
	DynAE	89.05	94.89	95.89
	DynRNN	73.79	81.47	85.19
	DynAERNN	77.76	85.77	91.55
	E-LSTM-D	82.97	88.45	98.09
GTA	DDNE	94.69	96.13	94.86
	DynAE	94.27	94.63	95.94
	DynRNN	77.62	79.61	85.46
	DynAERNN	79.81	85.59	94.33
	E-LSTM-D	84.50	89.76	98.17
RB	DDNE	89.58	94.52	94.43
	DynAE	73.18	92.46	94.01
	DynRNN	70.97	94.39	85.05
	DynAERNN	75.63	86.77	95.42
	E-LSTM-D	73.06	87.03	98.28
GB	DDNE	96.48	96.87	94.29
	DynAE	97.74	96.06	94.01
	DynRNN	84.27	91.95	85.49
	DynAERNN	93.36	89.33	95.38
	E-LSTM-D	85.78	86.21	98.24
Dyn-One	DDNE	96.65	96.69	93.92
	DynAE	98.00	95.38	94.12
	DynRNN	89.47	94.82	85.13
	DynAERNN	95.41	91.58	95.60
	E-LSTM-D	90.54	86.02	98.24
O	DDNE	97.93	97.14	94.29
	DynAE	98.95	96.87	94.31
	DynRNN	95.09	95.45	85.31
	DynAERNN	98.84	90.14	96.65
	E-LSTM-D	93.86	89.84	98.21

believe that our method aims at a target link, so the perturbation caused by the entire graph is imperceptible.

(2) *It is easier to attack a link as existence than to make it predict it as non-existence.* Although the trigger size of attack scenario II is smaller than attack scenario I, the attack effect becomes better in attack scenario II in terms of ASR, e.g., the ASR of DDNE reaches 100% on Radoslaw and the ASR of E-LSTM-D reaches 100% on Fb-forum. There are two possible reasons for this phenomenon. Firstly, we observed

that there is no interference from redundant neighbors between two nodes. This means that establishing a connection between the two nodes through a trigger is relatively easier. Secondly, when it comes to link prediction, the models tend to focus more on the existence state of the links. As a result, the attacker finds it easier to manipulate a link's existence compared to its non-existence.

7.5.6 Attack Transferability

In practical scenarios, attackers may not have access to detailed information about the target DLP model beforehand. Therefore, conducting a black-box attack, where no structural or parameter information of the DLP model is known, is more realistic. To evaluate the effectiveness of our method under the black-box setting, we use one DLP model as the attack discriminator. We then transfer the generated trigger to backdoor other DLP models, which are the target models. This type of attack is referred to as a transferable attack. Take one of the datasets as an example, Fig. 7.3 shows the transferability attack results on Radoslaw.

In attack scenario I, we find that our method's attack effect is significant against DDNE model, e.g., the ASR of attacking DDNE reaches 99.15%, using DynAE as the attack discriminator in Fig. 7.3. When attacking the DDNE model, ASR can reach more than 80%. This shows that DDNE has a strong ability to capture the network structure, and its robustness needs to be further strengthened.

In attack scenario II, our method can maintain good performance as well by achieving over 99% ASR against several models. We find that the attack effects on the

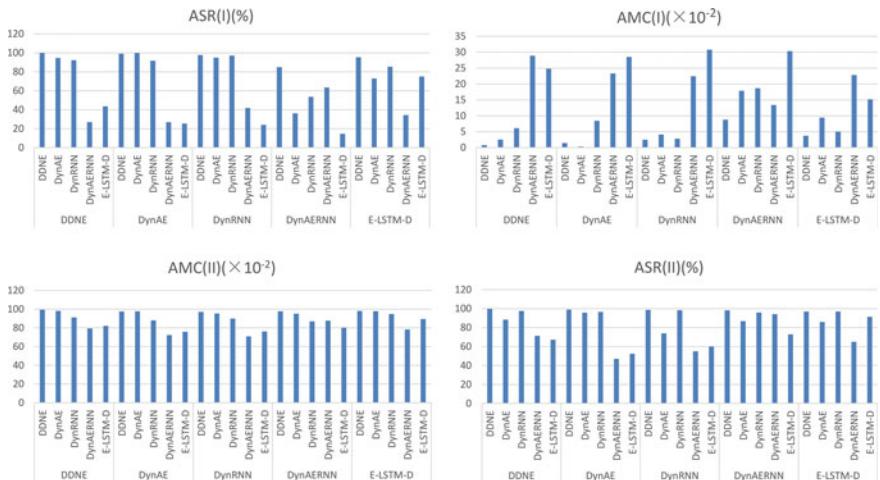


Fig. 7.3 The transferability of our method on the Radoslaw dataset. I and II are the results under attack scenario I and attack scenario II, respectively

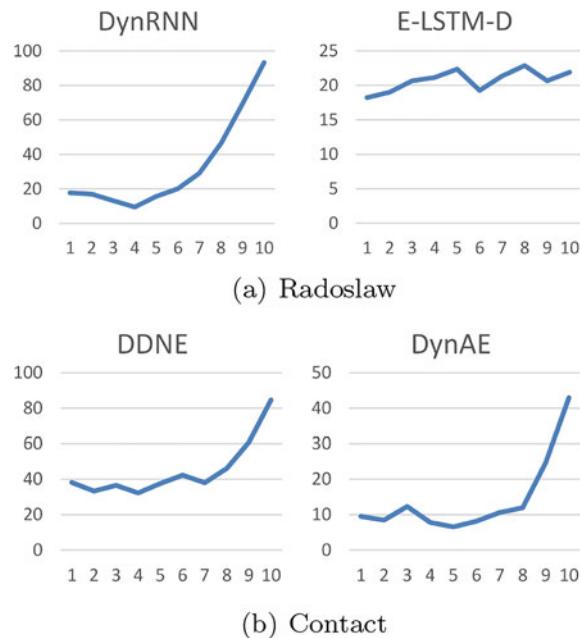
DDNE and DynAE models are similar, e.g., the ASR of DDNE achieves 99.08% and the ASR of DynAE achieves 95.68% in Fig. 7.3 on DynAE as attack discriminator. One possible reason is the similarity in the encoder and decoder structures used by the two models. Despite capturing feature information differently, most DLP methods can keep good performance. By selecting a high-performing DLP model as the attack discriminator, the attacker can effectively execute our method attack in the black-box setting.

7.5.7 Trigger Injection Timestamp Analysis

Dynamic networks have a temporal dimension that affects link prediction differently compared to static networks. To investigate the impact of dynamic link prediction on the attack at different timestamps, we conducted an experiment on the Radoslaw and Contact datasets. These datasets contain more timestamps for a sequence compared to other datasets, allowing us to better observe the effect of the attack at different points in time.

Figure 7.4 shows that the ASR varies greatly when the trigger injection timestamp is different, indicating that the attack needs to pay attention to the timestamp factor in order to launch an effective attack. Our method shows great differences at different timestamps. In Contact dataset and DDNE model, if the insert position of the trigger

Fig. 7.4 Different timestamps. 1–10 of the vertical axis means that the attacker manipulates the data at ten timestamps, and the rest only manipulates the data with the corresponding timestamp



is limited to the 4-th timestamp, the ASR is only 32.51%. If the insert position of the trigger is focused on the 10-th timestamp, the ASR can reach 84.98%. Experiments on other datasets and models find that the same characteristics exist, except for the E-LSTM-D model. Since the E-LSTM-D model directly splices the features of all timestamps together, the features of all timestamps are equally important. In particular, we observed that when the attack is focused on the most recent timestamp, ASR is the highest. This suggests that the timing of the attack is not only dependent on the model's performance but also closely related to the effectiveness of the attack. The temporal evolution of the network plays a crucial role in determining the success of the attack, highlighting the importance of considering the time dimension in dynamic link prediction attacks.

7.6 Conclusion

The main focus of this chapter is to introduce a backdoor attack framework called Dyn-Backdoor, specifically designed for Deep Learning-based Link Prediction (DLP). This framework utilizes Generative Adversarial Networks (GAN) and gradient exploration techniques to create a trigger. Through extensive experimentation, we demonstrate the effectiveness of our method in compromising the performance of DLP models. The dynamic nature of the model enables it to be trained on data containing the trigger, allowing the attacker to launch an attack by invoking the trigger.

References

- Yao, L., Wang, L., Pan, L., Yao, K.: Link prediction based on common-neighbors for dynamic social network. In: The 7th International Conference on Ambient Systems, Networks and Technologies (ANT 2016)/The 6th International Conference on Sustainable Energy Information Technology (SEIT-2016)/Affiliated Workshops, May 23–26, 2016, Madrid, Spain. Procedia Computer Science, vol. 83, pp. 82–89. Elsevier (2016)
- Zhang, Z., Wen, J., Sun, L., Deng, Q., Su, S., Yao, P.: Efficient incremental dynamic link prediction algorithms in social network. *Knowl. Based Syst.* **132**, 226–235 (2017)
- Acer, U.G., Drineas, P., Abouzeid, A.A.: Random walks in time-graphs. In: Proceedings of the Second International Workshop on Mobile Opportunistic Networking, MobiOpp '10, Pisa, Italy, February 22–23, 2010, pp. 93–100. ACM (2010)
- Ibrahim, N.M.A., Chen, L., Wang, Y., Li, B., Li, Y., Liu, W.: Sampling-based algorithm for link prediction in temporal networks. *Inf. Sci.* **374**, 1–14 (2016)
- Ibrahim, N.M.A., Chen, L.: An efficient algorithm for link prediction in temporal uncertain social networks. *Inf. Sci.* **331**, 120–136 (2016)
- Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: Companion of the Web Conference 2018 on the Web Conference 2018, WWW 2018, Lyon, France, April 23–27, 2018, pp. 969–976. ACM (2018)
- Goyal, P., Kamra, N., He, X., Liu, Y.: Dymgem: Deep embedding method for dynamic graphs. *CoRR* (2018). [arXiv:abs/1805.11273](https://arxiv.org/abs/1805.11273)

8. Goyal, P., Chhetri, S.R., Canedo, A.: dyngraph2vec: capturing network dynamics using dynamic graph representation learning. *Knowl. Based Syst.* **187** (2020)
9. Xuan, Q., Xiao, H., Fu, C., Liu, Y.: Evolving convolutional neural network and its application in fine-grained visual categorization. *IEEE Access* **6**, 31110–31116 (2018)
10. Li, T., Zhang, J., Yu, P.S., Zhang, Y., Yan, Y.: Deep dynamic network embedding for link prediction. *IEEE Access* **6**, 29219–29230 (2018)
11. Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T.B., Leiserson, C.E.: Evolvegcn: evolving graph convolutional networks for dynamic graphs. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020, pp. 5363–5370. AAAI Press (2020)
12. Chen, J., Lin, X., Jia, C., Li, Y., Liu, Y.: Generative dynamic link prediction. *Chaos* **29**(12), 123111 (2019)
13. Chen, J., Zhang, J., Xu, X., Fu, C., Zhang, D., Zhang, Q., Xuan, Q.: E-LSTM-D: a deep learning framework for dynamic network link prediction. *IEEE Trans. Syst. Man Cybern. Syst.* **51**(6), 3699–3712 (2021)
14. Chen, J., Zhang, J., Chen, Z., Du, M., Xuan, Q.: Time-aware gradient attack on dynamic network link prediction. *CoRR* (2019). [arXiv:abs/1911.10561](https://arxiv.org/abs/1911.10561)
15. Fan, H., Wang, B., Zhou, P., Li, A., Xu, Z., Fu, C., Li, H., Chen, Y.: Reinforcement learning-based black-box evasion attacks to link prediction in dynamic graphs. In: 2021 IEEE 23rd International Conference on High Performance Computing & Communications; 7th International Conference on Data Science & Systems; 19th International Conference on Smart City; 7th International Conference on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys), Haikou, Hainan, China, December 20–22, 2021, pp. 933–940. IEEE (2021)
16. Zhang, Z., Jia, J., Wang, B., Gong, N.Z.: Backdoor attacks to graph neural networks. In: SACMAT '21: The 26th ACM Symposium on Access Control Models and Technologies, Virtual Event, Spain, June 16–18, 2021, pp. 15–26. ACM (2021)
17. Xi, Z., Pang, R., Ji, S., Wang, T.: Graph backdoor. In: 30th USENIX Security Symposium, USENIX Security 2021, August 11–13, 2021, pp. 1523–1540. USENIX Association (2021)
18. Xu, J., Xue, M., Picek, S.: Explainability-based backdoor attacks against graph neural networks. In: WiseML@WiSec 2021: Proceedings of the 3rd ACM Workshop on Wireless Security and Machine Learning, Abu Dhabi, United Arab Emirates, July 2, 2021, pp. 31–36. ACM (2021)
19. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8–13 2014, Montreal, Quebec, Canada, pp. 2672–2680 (2014)
20. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
21. Li, X., Du, N., Li, H., Li, K., Gao, J., Zhang, A.: A deep learning approach to link prediction in dynamic networks. In: Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24–26, 2014, pp. 289–297. SIAM (2014)
22. Li, T., Wang, B., Jiang, Y., Zhang, Y., Yan, Y.: Restricted boltzmann machine-based approaches for link prediction in dynamic networks. *IEEE Access* **6**, 29940–29951 (2018)
23. Selvarajah, K., Ragunathan, K., Kobti, Z., Kargar, M.: Dynamic network link prediction by learning effective subgraphs using CNN-LSTM. In: 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19–24, 2020, pp. 1–8. IEEE (2020)
24. Chen, J., Wang, X., Xu, X.: GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction. *Appl. Intell.* **52**(7), 7513–7528 (2022)
25. Liu, J., Xu, C., Yin, C., Wu, W., Song, Y.: K-core based temporal graph convolutional network for dynamic graphs. *IEEE Trans. Knowl. Data Eng.* **34**(8), 3841–3853 (2022)

26. Liu, K., Li, Y., Yang, J., Liu, Y., Yao, Y.: Generative principal component thermography for enhanced defect detection and analysis. *IEEE Trans. Instrum. Meas.* **69**(10), 8261–8269 (2020)
27. Liu, K., Tang, Y., Lou, W., Liu, Y., Yang, J., Yao, Y.: A thermographic data augmentation and signal separation method for defect detection. *Meas. Sci. Technol.* **32**(4), 045401 (10pp) (2021)
28. Lei, K., Qin, M., Bai, B., Zhang, G., Yang, M.: GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks. In: 2019 IEEE Conference on Computer Communications, INFOCOM 2019, Paris, France, April 29–May 2, 2019, pp. 388–396. IEEE (2019)
29. Yang, M., Liu, J., Chen, L., Zhao, Z., Chen, X., Shen, Y.: An advanced deep generative framework for temporal link prediction in dynamic networks. *IEEE Trans. Cybern.* **50**(12), 4946–4957 (2020)
30. Brand, M.: Fast low-rank modifications of the thin singular value decomposition. *Linear Algebr. Its Appl.* **415**(1), 20–30 (2006)
31. Zhang, Z., Cui, P., Pei, J., Wang, X., Zhu, W.: TIMERS: error-bounded SVD restart on dynamic networks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, pp. 224–231. AAAI Press (2018)
32. Li, J., Dani, H., Hu, X., Tang, J., Chang, Y., Liu, H.: Attributed network embedding for learning in a dynamic environment. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06–10, 2017, pp. 387–396. ACM (2017)
33. Ma, X., Sun, P., Wang, Y.: Graph regularized nonnegative matrix factorization for temporal link prediction in dynamic networks. *Phys. A* **496**, 121–136 (2018)
34. Zuo, Y., Liu, G., Lin, H., Guo, J., Hu, X., Wu, J.: Embedding temporal network via neighborhood formation. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19–23, 2018, pp. 2857–2866. ACM (2018)
35. Takahashi, T.: Indirect adversarial attacks via poisoning neighbors for graph convolutional networks. In: 2019 IEEE International Conference on Big Data (IEEE BigData), Los Angeles, CA, USA, December 9–12, 2019, pp. 1395–1400. IEEE (2019)
36. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: online learning of social representations. In: The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA—August 24–27, 2014, pp. 701–710. ACM (2014)
37. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13–17, 2016, pp. 855–864. ACM (2016)

Chapter 8

Attention Mechanism-Based Adversarial Attack Against DRL



8.1 Introduction

Reinforcement learning (RL) [1] is a subset of machine learning that is generally a continuous decision-making process. Deep Reinforcement Learning (DRL) makes full advantage of neural networks as parametric structures to optimize RL algorithms by integrating decision-making ability with deep learning perceptual ability. Convolutional neural networks have a naturally advantageous role in image processing, so DRL's strategy is optimized through neural network training. Recently, DRL has been developed rapidly in many areas, and has achieved significant results in the fields of game players, automatic driving, robot control, traffic control, financial transactions, and cybersecurity.

However, neural networks and deep learning models are vulnerable to adversarial attacks caused by maliciously created examples, which increases the pitfalls of RL. Zegedy et al. [2] first discovered that deep neural networks are extremely susceptible to attacks in the field of image classification. The fragility of neural networks is also explained by Goodfellow et al. [3], who suggested that the primary reason for the vulnerability of neural networks to adversarial perturbations lies in the nature of their linear behavior in high-dimensional spaces. Therefore, it will be a meaningful research on how to build a secure RL system.

The vulnerability of RL is also easily utilized by attackers. Existing attack methods can be classified into white-box and black-box attacks.

White-box attack was proposed earlier by Goodfellow [3], who first utilized gradient-based attacks in neural networks. Tretschk et al. [4] utilized inverse transformation networks to create perturbations in order to attack the RL environment. Then Huang et al. [5] adapted the FGSM attack method into the Atari2600 game. The white-box attack methodology of RL also includes start point-based adversarial attack on Q-learning (SPA) [6] and white-box-based adversarial attack on DQN(WBA) [7]. One of the effective attacks is the commonly dominant adversarial paradigm generation method (CDG) [8] and snooping threat models [9] attack model, which launches a destructive attack on the environment of DQN and PPO by observing the return

of the agent’s action. There is also a white-box attack method directed by the value function [10] similar to the strategic time attack idea of Lin et al. [11], which uses a value function-based model to evaluate the value of the current state to choose when to attack. Besides, poison attacks [6] are used in a multi-agent game environment on the MuJoCo platform.

On the other hand, black-box attack methods against RL have also been presented, such as policy induction attack (PIA) [12], strategic time attack, and puzzle attack methods presented by Lin et al. [13], and transferable black-box attack methods presented by Zhao et al. [14], which decrease the cumulative reward. Existing attack methods against RL agents suppose that the adversary can obtain the learning parameters of the target agent or the environment in which the agent interacts, and then launch a destructive attack on the DRL environment by observing the agent’s behavioral reward signals. In this chapter, we present a new attack method based on the above RL security research.

The main contribution of this chapter is to present a new DRL attack methodology.

- (1) Attackers can use the attention mechanism to obtain the complete model information and extract the easily disturbed pixel information from the feature information of the deep convolutional layer to improve the effectiveness of the attack.
- (2) In this chapter, we will integrate bicubic interpolation in DRL feature transformation with feature pixel reconstruction.
- (3) In experiment, we use DQN in the Flappybird [15] game environment to train RL agent.
- (4) After the attack, the bonus rewards for the duration of the game attack are reduced.

In addition, a perturbation size imperceptible to the human eye is used to better confuse the agent. The attack method presented in this chapter can produce a great deal of destructive power with a small perturbation magnitude.

8.2 Related Works

In this section, we describe the attack methods in the domain of reinforcement learning security.

DRL has the advantages of both deep learning and RL, while inheriting the shortcomings of both. Deep learning has long been shown to be vulnerable to malicious perturbations by attackers. In addition, network models are very fragile.

Huang et al. [5] first found that neural network policies are vulnerable to adversarial attacks in deep reinforcement learning. The attack they used was to add small perturbations to the input states of the DQN [16, 17] leading to erroneous actions of the agent, which is the FGSM attack method proposed by Goodfellow et al. [3]. Using this method, Huang et al. verified for the first time that the agents trained by DRL model based on DQN, TRPO algorithm [4, 10] are vulnerable to adversarial

perturbation attacks. Furthermore, they found that the transferability across datasets proposed by Szegedy et al. [2] in 2013 was also applied to RL. Currently, one of the most intensively researched application environments for DRL is the gaming environment. Lin et al. [13] proposed a black-box attack method based on Atari2600 game environments, which was strategic time attack. An obfuscation attack was also proposed by Lin et al. The experiment used the adversarial example generation algorithm proposed by Carlini and Wagner [18], which aims to start from a state at a certain moment in time and add a perturbation to the agent so that it reaches the desired state after n steps. Subsequently, Kos et al. [19] proposed an attack method guided by value function, whose main idea is to utilize the value function module to choose whether or not to attack the value of the current state. Behzadan and Munir [12] proposed PIA, which produces a perturbation at each time step to influence the next state. This attack method utilizes imitation learning to accelerate the creation of equivalent models, providing a new scheme for attacking DRL models in a black-box environment.

RL is also widely used for path planning. For example, Liu et al. [20] proposed an RL path planning attack method for base-valued iterative networks. Xiang et al. [21] proposed a counterattack method named SPA based on starting point attack, which is based on q-learning algorithm to train RL with the aim of building a probabilistic output model of path points based on the original model. Bai et al. [7] proposed WBA method for SPA attack methods based on DQN training, which requires not only angular analysis of points on the planning path, but is also restricted by the scenario. Chen et al. [8] proposed a new gradient-based attack method for the A3C algorithm in atrai game path planning scenarios. Chen also tried to find the main gradient perturbation zones and adds baffles to the optimized paths trained by the agent to trick the agent. Although this attack maximizes the agent's loss function for a short period of time and has some impact on its performance, the agent can still recover after a long period of training.

Besides the path planning problem, RL attack methods have been applied in other areas. Tretschk et al. [22] integrated new counterattack techniques from the anti-transformation network into the strategic network structure. Thus, through a series of attacks, the target policy network is able to optimize the adversarial rewards over the raw rewards during training. Behzadan and Munir [12] proposed a new black-box attack method based on the FGSM proposed by Huang et al. [5]. Using the multi-robot game in MuJoCo as the RL training environment and the game trained by Bansal et al. [6] as the evaluation environment. Russo et al. [23] proposed an attack method based on RL optimization. Matthew Inkawich et al. [9] presented SRA attack model. In addition, Zhao et al. [14] used DQN to train the seq2seq model and directly transferred the adversarial instances from the training model to A2C.

8.3 Preliminaries

In this section, we introduce the RL's basic model, the principle of attention mechanism, and the feature transformation method of RL neural networks, which will be used throughout the chapter.

8.3.1 The Basic Model of RL

Markov decision process (MDP) is the RL's theoretical framework, the next state is only related to the current state and is independent of the previous state. MDP is described by the tuples of (S, A, P, R, γ) , where $S = \{s_1, s_2, s_3, \dots, s_t\}$ is the state set, $A = \{a_1, a_2, a_3, \dots, a_t\}$ is the action set, P is the state transition probability, R is the reward function, and λ is the discount factor used to calculate the long-term cumulative return. The state transition matrix of Markov decision process is $P_{ss'}^a = P [S_{t+1} = s' | S_t = s, A_t = a]$. The goal of RL is to find the best policy in a given Markov decision process. The policy represented by π refers to the mapping of state set to action set, which means that the output distribution of actions in a given state s represented as $\pi(a|s) = p [A_t = a | S_t = s]$. When the agent employs this policy, the cumulative returns follow an arbitrary distribution.

Cumulative return is $G_t = R_{t+1} + \gamma R_{t+2} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$ used to measure

the value of the state s_i . However, the cumulative return is a random variable rather than a definite value and therefore cannot be characterized. However, its expectation is a definite value that can be defined as a function of the state value.

The expected value of the state-value function and the state-behavior value function are calculated by the Bellman equation. It can be obtained from the definition of the state-value function $v(s) = E[G_t | S_t = s] = E[R_{t+1} + \gamma v(S_{t+1}) | S_t = s]$. The state-action value function also can be obtained as $q_{\pi}(s, a) = E_{\pi}[R_{t+1} + \gamma q(S_{t+1}, A_{t+1}) | S_t = s, A_t = a]$. If the value function of the optimal state action is known, the optimal policy can be determined by direct maximization.

8.3.2 Attention Mechanism

Attention models are widely used in deep learning tasks in natural language processing, image recognition, speech recognition, and other fields. It is one of the core technologies of deep learning technology and deserves more attention and deeper understanding. Attentional mechanisms have their roots in the study of human vision. In cognitive science, due to bottlenecks in information processing, humans selectively focus on an important part of all information while ignoring other visible information, which is often referred to as an attentional mechanism. The attention mechanism is

concerned with the assignment of input weights. It is first used in encoder-decoders, where the attention mechanism gets the input variables of the next layer by weighted averaging the hidden states of all time steps of the encoder. Attention mechanism was first proposed by Bahdanau et al. [24], which is primarily used as a translation model to address translation kernel alignment issues (seq2seq+attention is adopted in this chapter). In this chapter, we will extract the depth features of the model, reconstruct the features using the depth feature map, and fuse them with the input states to obtain the attention feature $W_{s'}$, and extract the pixel features $W_{s''}$ from $W_{s'}$, thereby confusing the agent.

8.3.3 DRL Feature Transformation

We extracted the depth feature map of DRL and performed the feature transform. The feature transform of DRL is coupled with the bicubic interpolation of the image. The computational process of bicubic interpolation is more complex compared to nearest neighbor interpolation and bilinear interpolation. Bicubic interpolation is the most commonly used interpolation method in two dimensions, and is used here to obtain the value of the function f at points (x, y) by weighted average of the last 16 sampling points in the rectangular grid. In addition, we need to perform interpolation of the cubic function using two polynomials, one in each direction. We combine the depth features of the DRL with the pixel transformation method to obtain the reconstructed features of the depth feature map.

8.4 Methodology

In our work, we use an attention-based attack approach to perform a white-box attack on a trained modeling strategy π_θ , where the attacker has access to the model structure and parameters. First, we give a general block diagram of the attack, as shown in Fig. 8.1. In any model of the threat, we assume that the adversary can intercept and manipulate incoming game frames. In our threat model, the adversary can spy on rewards and action signals.

8.4.1 General Introduction to Attack

Vulnerability analysis of RL: The learning process of RL agents is itself a continuous game, and even a small perturbation may break the equilibrium that is about to be established. As we all know, neural networks, as the central technology of deep learning, have been proved to be extremely vulnerable to interference, on which many scholars in different fields have expressed their opinions and proposed many

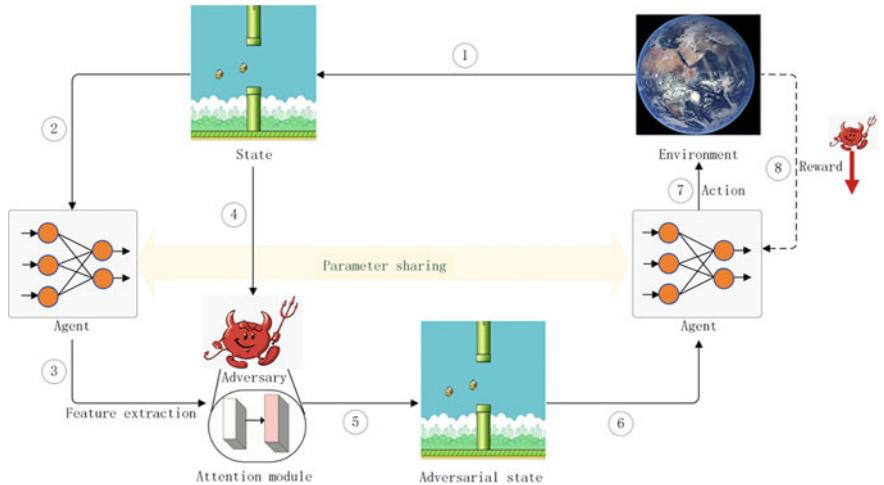


Fig. 8.1 Framework of attention threat model for DRL

defense methods. DRL utilizes not only deep learning but also its own policy learning, making it more vulnerable to attacks. Not only that, the standard reinforcement learning evaluation method has not been explicitly proposed. Currently, the main evaluation criteria are observing the reward value and visualizing the training effect, but the criteria for calculating the reward value also changes with the environment and training method. In addition, the reward values during training are in a dynamic computation phase, which will inevitably lead to fitting or underfitting problems. In short, reinforcement learning is still vulnerable. In this chapter, we propose a new attack method that can break the equilibrium state of a trained DQN during testing. Meanwhile, the long-term cumulative reward value is still used as an index to evaluate the effectiveness of the attack method. We use several previously proposed attacks to compare with our proposed method and compare the reward values for the same perturbation size to understand the successful effect of our proposed method.

Object of attack: In this chapter, we will attack models trained based on the DQN algorithm. The non-policy applied in Q-Learning is extended in DQN. The major difference compared to reinforcement learning based on the traditional Q-Learning algorithm is that the DQN model no longer generates a complete Q-table at initialization. By inputting the features of the current state, the neural network generates the Q-value of the state. We then choose an action based on the Q-value. However, the presence of the neural network makes the DRL more susceptible to interference. We will attack the trained DQN during testing. To see if our attack method still has limitations, we compare it using several previously proposed attack methods. We compare the size of the reward values for the same perturbation size and observe the loss of the predicted Q-value and the target Q-value. Since we performed the attacks during testing, the corresponding loss after the attack is not as large as possible. Instead, we want it to be as small as possible, i.e., the smaller the value of

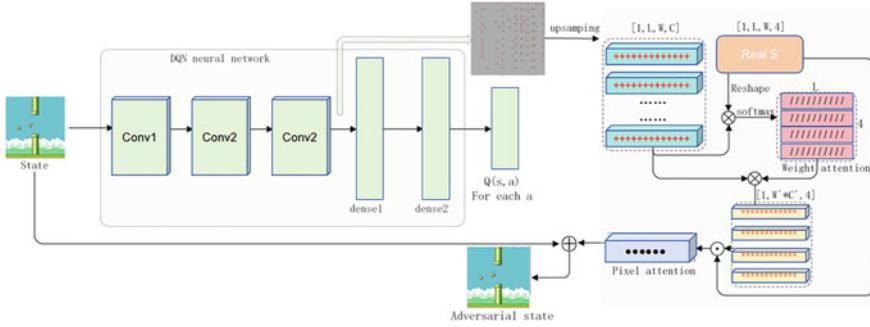


Fig. 8.2 Framework of attention mechanism method

our loss in the case of an attack, the better. But it is not done by a specific operation, which improves the effectiveness of the attack of our method.

Adversarial attack based on attention mechanism: In neural networks, the shallow features of the model preserve most of the features of the input samples, and the deep features have a larger field of view focusing on features that are significant for sample identification and processing. However, it is difficult for the human visual system to predict the original sample by such deep features. We use deep pixel features and weighted attention along with bicubic interpolation to generate adversarial perturbations. The effectiveness of a perturbation is measured by the attack effect and the intensity of the perturbation, which depends on the size and distribution of the perturbation. In general, the larger the perturbation and the wider the distribution, the better the attack effect. However, it would be better to obtain the same attack effect by a smaller perturbation and a narrower distribution. Based on above, we apply a new attack method in RL which is the adversarial attack method based on attention mechanism. As shown in Fig. 8.2, the general framework of the attack method based on the attention mechanism is given. As can be seen from the figure, we choose the output of the second convolutional layer of the DQN-based neural network model as the object of attention. Then, we extract the output feature values of this layer and perform the attention transformation of weights and pixels. Also, we perform feature reconstruction and size transformation. Finally, the perturbation values are obtained and added to the clean state to generate the adversarial example.

8.4.2 Attack Method Description

In our experiments, we extract deep features of the model, reconstruct the features using deep feature maps, and fuse them with the input state to confuse the agent. The first step in our attack method is to extract the features of the deep neural network. During testing, the size of the state is $[L, W, C]$. The input to the DQN algorithm for the model is four consecutive image frames. Thus the input size of the model is

$[L, W, C, D]$. Here, W is the number of pixels in the vertical direction of the state, L is the number of pixels in the horizontal direction of the state, C is the number of pixel channels, and D is the number of iterative states. We extract the deep feature image g with the size of $[1, L_1, W_1, C_1]$. In addition, the feature map needs to be enriched using bicubic interpolation to obtain a reconstructed feature image of size $[1, L, W, C]$. Next, we need to assign weights. First of all, we need to extract the original state $S^r e$ with the size of $[L, W, C, 4]$. Then turning the measure to the size of $[L, 4, B]$ by reshaping, where $B = W * C$. Next we change the figure size $[1, L, W, C]$ for reconstructing features g_s . Through the reshaping conversion, we gain the reconstructed feature graph g_s , whose size is $[1, C, B_s]$. Then calculate the weight of channel attention. The calculation formula is as follows:

$$W_{rle} = \text{softmax}(\tanh(s^{re} \otimes g_s)), \quad (8.1)$$

where $\text{softmax}(.)$ is activation function. We also need to reshape the feature of W_{rle} to gain the new size of $[1, 1, L, 4]$. What's more, it is necessary to obtain the weight of reconstructed channel-spatial attention.

$$W_{rle}^{re} = W_{rle} \otimes g_m, \quad (8.2)$$

where W_{rle} is the weight of channel-spatial attention, g_m is the feature graph after the transformation of original state space, and then further extract the deep features. First of all, it is necessary to change the size of $W_{rle}^r e = W_{rle}$ into $[1, B, 4]$. At the same time, the original state transition size needs to reshape into $[L, B_s, 4]$. After converting above two, activate by the function $\tanh(.)$ and then get the final attentional feature:

$$W_{rle} = \text{softmax}(\tanh(vec)) \quad \text{where } vec = \frac{\sum_{i=1}^n Z_i}{n} \quad (8.3)$$

$$x = W_{rlc}^{re*} \otimes S^r e, \quad (8.4)$$

where Z_i is the average value of the second-dimensional elements of x . Finally, adjust the size of W_{att} into $[1, L, W, 1]$ to get the mapping feature W_{adv} , which is the perturbation we will use. During testing, the perturbation is obtained dynamically. Running every frame of image to get a deep perturbation $\rho = W_{adv}$ based on attention, which is added to the input state to get the adversarial state:

$$s_{adv}^t = s_{re}^t + \rho^t, \quad (8.5)$$

where s_{adv}^t is the perturbation state at time t , s_{re}^t is the original state at time t , and ρ^t is the perturbation used to generate the perturbation state at time t .

8.5 Experimental Evaluation

In this section, we give the results of the baseline methods and our proposed attack method. We also analyze the data and curves of the experiment results.

8.5.1 Experimental Description

Experimental environment: Flappybird [15] game environment based on DQN [16, 17] deep reinforcement learning.

The perturbation limits: The common perturbation calculation methods include L_0 norm, L_2 norm, and L_∞ norm, where norm is used to count the number of changes in pixel points. L_∞ norm is used to calculate the pixel with the most perturbation. L_∞ norm is similar to Euclidean distance and calculates the root mean square of the sum of the squares of the absolute values of the disturbed pixel points.

Baseline attack method: FGSM [3], PGD [25].

We perform comparative experiments using gradient-based adversarial attack methods. During testing, we use the cross entropy between the predicted Q-value and the target Q-value as the loss function. Then the gradient value of the loss function is added as a perturbation in the input state.

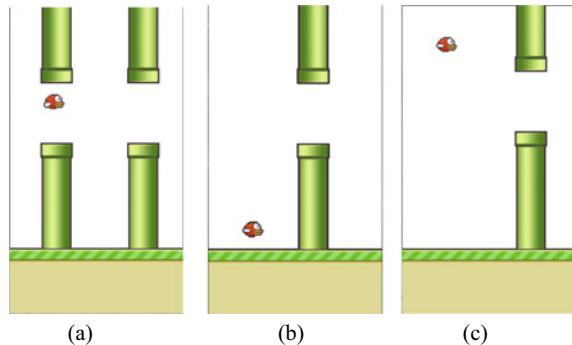
8.5.2 Experimental Results

The experimental environment is Flappybird game environment based on DQN algorithm. We propose an attacking method based on attentional mechanism, named RLAT, which centers on using the deep features of neural networks to generate perturbed pixel matrices for the attack.

As a visual comparison, we show three screenshots of the game environment under different scenarios. As shown in Fig. 8.3, we can see three states under different attack methods, including the undisturbed state, the state under the RLAT method, and the state under the baseline attack method. Figure 8.3a shows the path of the undisturbed agent. Figure 8.3b uses the attentional mechanism to attack. Under the same size of perturbation, the agent's action deviates significantly from the actual situation. As shown in Fig. 8.3c, the motion direction of the attacked bird is consistent with the baseline. Obviously, the bird is still aware of the upward movement, so the reward value must be higher than our attack method.

The development of computer vision provides the RL model to realize the expected results. And the computer's observation of the environment also greatly contributes to the development of neural networks. In addition, the RL model based on the DQN algorithm makes full use of convolutional neural networks. Each layer of the convolution is associated with different information about the input state. The

Fig. 8.3 Agent action visualization diagram



attention of the visual neural network is very helpful because it helps us to understand if the network is viewing the appropriate part of the image or if the network is producing misdirection. In our experiments, we pay more attention to the convolution of the last layer to get its gradient. Then the gradient of the feature map is averaged. Finally, we take out the activation values of the last convolutional layer. At the same time, we multiply them by the average of the feature gradients. This process can be understood as multiplying the importance of each channel by the convolutional activation value instead of performing a weighting operation. Finally, we generate a heat map based on the obtained values and then merge it with the original image. As shown in Fig. 8.4, it is a graph of the state gradient distribution generated based on the various attack methods. The first three maps correlate to the four different attack approaches, while Fig. 8.4a shows the unattacked state. Figure 8.5 is the superposition of Fig. 8.4 states and corresponding heat maps. It can be seen from Fig. 8.5 that the neural network can focus on the aggregation part of the image, which is around the bird, other than that the main portion remains the location of the bird and the pipeline portion.

The perturbation in the experiment is calculated using l_2 norm. FGSM and PGD are used in the comparative experiment. The experimental results are not compared with the results under the minimum perturbation, but the l_2 perturbation norm is restricted to (1.15 ± 0.1) . The comparative experimental results of several attack methods under the same perturbation size are given below. In the experiments, we compare the results under large perturbations, where the choice of perturbation size is also highly destructive. As shown in Fig. 8.6, the box plots of reward values corresponding to different attack methods are given in the figure. As shown in Table 8.1, the table gives the average of the reward values corresponding to the different attack methods, where none means no attack. The second column represents the attack method, the third column represents the average reward value for rounds 0 to 200, and the fourth column represents the average reward value for rounds 200 to 400. As can be seen from Table 8.1, the reward value obtained by the RLAT attack method is smaller than the other attack methods, and the upper limit of the reward value is also smaller than the corresponding reward value of other attack methods. Therefore, under large perturbation, our attack method is more advantageous than other attack methods.

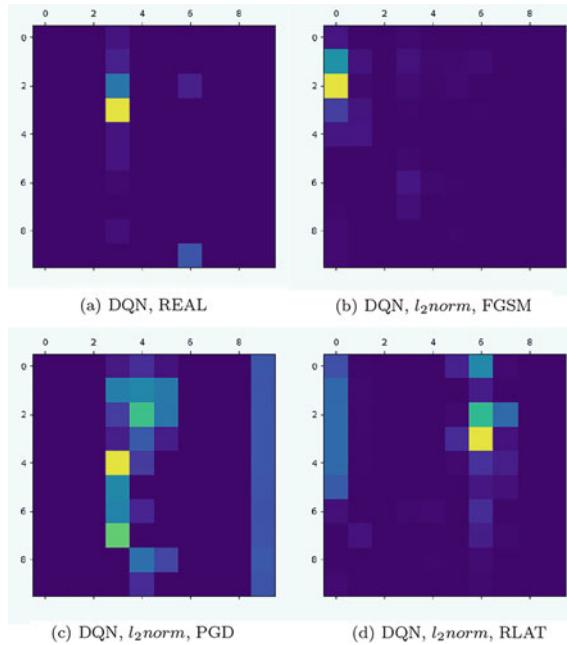


Fig. 8.4 Hot map of pixel gradient distribution. The last three figures, respectively, correspond to different attack methods. The perturbation size is limited to the same range ($l_2(\rho) = 1.15 \pm 0.1$), which are the visual results at the end of the game after the attack. The first diagram is the undisturbed heat map

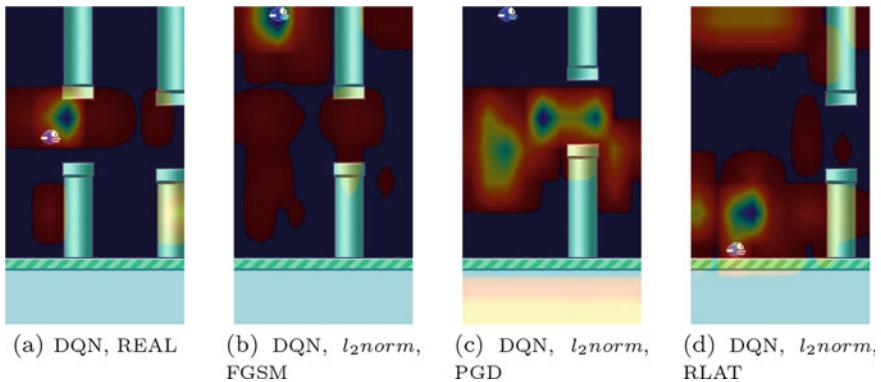


Fig. 8.5 Visualization results after merging heat map with original image. The last three figures correspond to three different attack methods, respectively, and the perturbation size is limited to the same range ($l_2(\rho) = 1.15 \pm 0.1$), which is the visualization result when the game ends. The first figure is the heat map without perturbations. The figure is the superposition of the heat map corresponding states in Fig. 8.5

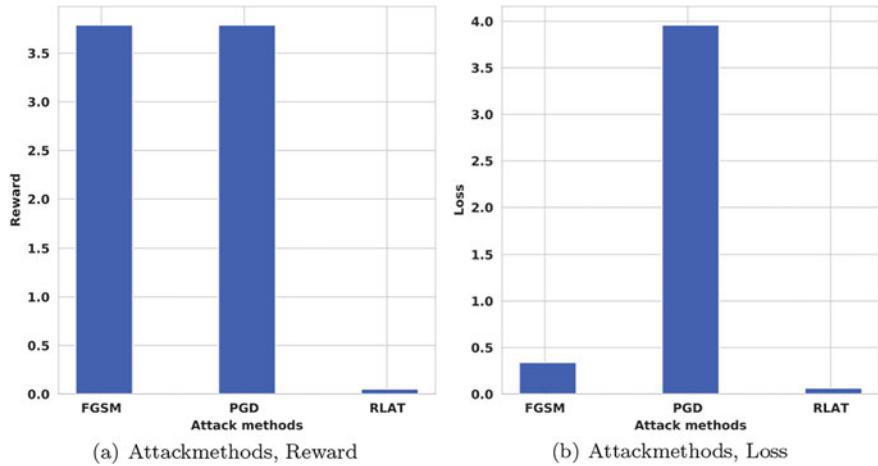


Fig. 8.6 Rewards and loss values under different attack methods

Table 8.1 Reward results under different attack methods

Game	Test attack $l_2((\rho)) = 1.15 \pm 0.1$	Reward (0–200 rounds mean)	Reward (200–400 rounds mean)
	None	438	500
Flappybird	FGSM	3.79	3.79
	PGD	3.79	3.79
	RLAT	0.052	0.052

Since the attack is executed during testing, the attacker uses a different loss gradient than the one used during model training. Therefore, we don't need to update the model parameters and only use the trained model to obtain the predicted Q-value. Meanwhile, we use the reward value strategy of the current state to obtain the maximum target Q-value corresponding to the next state, and the gradient used by the gradient-based attack method is the input gradient value corresponding to the loss of the predicted Q-value and the target Q-value. In the experiment, we also record the loss value between the predicted and target Q-value at the end of each round of the game. As shown in Table 8.2, a box plot of the loss values corresponding to the different attack methods is given in the table, where none indicates no attack, the third column represents the average of the loss values from rounds 0 to 200, and the fourth column represents the average of the loss values from rounds 200 to 400. From the table, we can clearly see the upper and lower limits of the loss values, as well as some abnormal loss values. As can be seen from Table 8.2, the RLAT attack method corresponds to a smaller loss of values compared to the other attack methods. After that game environment is attacked, it is difficult for both the observer and the agent to know whether the environment is maliciously interfered or not. Among the other attack methods, the FGSM attack method is more advantageous. So, it can be concluded that the attack method of RLAT and FGSM has a better ability to confuse

Table 8.2 Loss results under different attack methods

Game	Test attack $l_2((\rho)) = 1.15 \pm 0.1$	Loss (0–200 rounds mean)	Loss (200–400 rounds mean)
	None	0.12	0.2265
Flappybird	FGSM	0.304	0.372
	PGD	3.8	3.87
	RLAT	0.013	0.014

agents. More importantly, our attack method is better than FGSM. The greater loss is only the action of the bird before it crosses the barrier, which has little effect on the failure of the game. The reward value of the state crossing the barrier is very small, while the reward value of the attacked state is relatively large. Therefore, under the large perturbation condition, our attack method is more likely to confuse the agent than other attack methods. According to the above experiments, we organized and analyzed the data during the experiments. The experimental results are summarized as shown in Fig. 8.6. Figure 8.6a represents the average reward value obtained according to the corresponding attack method, and the loss calculated in Fig. 8.6b is the cross-entropy loss between the predicted Q-value and the target Q-value in the experiment. RLAT refers to the new RL attack method proposed in this chapter.

8.6 Conclusion

Through the above experimental analysis, we have investigated the effect of deep features of DRL neural network models on DRL environment observation. According to the characteristics of the adversarial attack, we propose a new attack method, which no longer needs to obtain the gradient information of the target model through backpropagation, but only needs to extract the deep features and perform the feature transformation, and finally get the perturbation, so as to effectively confuse the agent. Similarly, by adding perturbations to the environment state of the DRL, we make the reward value obtained by each action of the agent smaller and increase the frequency of game failure, which essentially changes the strategy of the agent.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement learning: an introduction. MIT press (2018)
2. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). [arXiv:1312.6199](https://arxiv.org/abs/1312.6199)
3. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). [arXiv:1412.6572](https://arxiv.org/abs/1412.6572)
4. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning, pp. 1928–1937 (2016)

5. Huang, S., Papernot, N., Goodfellow, I., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies (2017). [arXiv:1702.02284](https://arxiv.org/abs/1702.02284)
6. Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., Mordatch, I.: Emergent complexity via multi-agent competition (2017). [arXiv:1710.03748](https://arxiv.org/abs/1710.03748)
7. Bai, X., Niu, W., Liu, J., Gao, X., Xiang, Y., Liu, J.: Adversarial examples construction towards white-box q table variation in dqn pathfinding training. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), pp. 781–787. IEEE (2018)
8. Chen, T., Niu, W., Xiang, Y., Bai, X., Liu, J., Han, Z., Li, G.: Gradient band-based adversarial training for generalized attack immunity of a3c path finding (2018). [arXiv:1807.06752](https://arxiv.org/abs/1807.06752)
9. Inkawich, M., Chen, Y., Li, H.: Snooping attacks on deep reinforcement learning (2019). [arXiv:1905.11832](https://arxiv.org/abs/1905.11832)
10. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897 (2015)
11. Lin, Y.C., Liu, M.Y., Sun, M., Huang, J.B.: Detecting adversarial attacks on neural network policies with visual foresight (2017). [arXiv:1710.00814](https://arxiv.org/abs/1710.00814)
12. Behzadan, V., Munir, A.: Vulnerability of deep reinforcement learning to policy induction attacks. In: International Conference on Machine Learning and Data Mining in Pattern Recognition, pp. 262–275. Springer, Berlin (2017)
13. Yang, Y., Xiang, Y., Han, L.: A method to effectively detect vulnerabilities on path planning of vin. In: Information and Communications Security: 19th International Conference, ICICS 2017, Beijing, China, December 6–8, 2017, Proceedings, vol. 10631, p. 374. Springer, Berlin (2018)
14. Zhao, Y., Shumailov, I., Cui, H., Gao, X., Mullins, R., Anderson, R.: Blackbox attacks on reinforcement learning agents using approximated temporal information (2019). [arXiv:1909.02918](https://arxiv.org/abs/1909.02918)
15. Ebeling-Rump, M., Kao, M., Hervieux-Moore, Z.: Applying q-learning to flappy bird. Queen's University, Department Of Mathematics And Statistics (2016)
16. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013). [arXiv:1312.5602](https://arxiv.org/abs/1312.5602)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015)
18. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (sp), pp. 39–57. IEEE (2017)
19. Kos, J., Song, D.: Delving into adversarial attacks on deep policies (2017). [arXiv:1705.06452](https://arxiv.org/abs/1705.06452)
20. Lin, Y.C., Hong, Z.W., Liao, Y.H., Shih, M.L., Liu, M.Y., Sun, M.: Tactics of adversarial attack on deep reinforcement learning agents (2017). [arXiv:1703.06748](https://arxiv.org/abs/1703.06748)
21. Xiang, Y., Niu, W., Liu, J., Chen, T., Han, Z.: A pca-based model to predict adversarial examples on q-learning of path finding. In: 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC), pp. 773–780. IEEE (2018)
22. Tretschk, E., Oh, S.J., Fritz, M.: Sequential attacks on agents for long-term adversarial goals (2018). [arXiv:1805.12487](https://arxiv.org/abs/1805.12487)
23. Russo, A., Proutiere, A.: Optimal attacks on reinforcement learning policies (2019). [arXiv:1907.13548](https://arxiv.org/abs/1907.13548)
24. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014). [arXiv:1409.0473](https://arxiv.org/abs/1409.0473)
25. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks (2017). [arXiv:1706.06083](https://arxiv.org/abs/1706.06083)

Part II

Defenses for Deep Learning

Effective defense methods are crucial for the application of deep learning in sensitive security fields. Existing defense methods can be categorized into three types based on their defensive nature: data modification defense methods, model modification defense methods, and auxiliary detector defense methods. Data modification defense methods defend against adversarial attacks by fine-tuning input data and filtering out adversarial perturbations. Model modification defense methods focus on adjusting neurons and model frameworks from within the model to achieve defense against adversarial attacks. Auxiliary detector defense methods train adversarial sample detectors to prevent adversarial samples from being input into deep learning models, achieving defense against adversarial attacks.

Chapter 9 introduces a local gradient detection defense method, which belongs to the category of auxiliary detector defense methods. By calculating the local gradients of some benign examples and misclassified noisy examples, this method can accurately distinguish between adversarial examples, misclassified natural inputs, and benign examples. Through extensive experiments, it is shown that this method outperforms state-of-the-art (SOTA) baselines.

Chapter 10 presents a method that focuses the model's attention on key feature regions and enhances target contours. This method belongs to the category of data modification defense methods. First, adversarial examples are generated using various attack methods such as FGSM and MI-FGSM. Then, the adversarial samples are input into the target DNN. Subsequently, feature maps are created based on the shallow layers of the target DNN, and features are extracted.

Chapter 11 introduces a defense method for unknown attacks, which belongs to the category of model modification defense methods. First, the influence of neurons in selected layers is calculated using several batches of benign examples. Then, front neurons that may be exploited by an attacker and tail neurons are identified. Next, the front neurons are strengthened and the tail neurons are suppressed to generate counteracting perturbations that can mitigate and offset general adversarial perturbations. This method constructs an attack-agnostic defense without any prior knowledge of specific attack information and can serve as a general tool.

Chapter 12 presents a lightweight detector based on adaptive channel transformation, which belongs to the category of auxiliary detector defense methods. The optimal type of channel transformation and the minimum number of channel transformations are selected using a cuckoo search algorithm. This method not only can detect both adversarial and clean samples but also can identify attack types such as white-box attacks and black-box attacks.

Chapter 13 introduces a defense method based on weight evolution frequency, which belongs to the category of model modification defense methods. This method defines the concept of Weight Evolution Frequency Matrix (WEF-Matrix) to record the weight evolution frequency of the penultimate layer of the model and upload it to the server. The server can then differentiate between freeloaders and benign clients based on Euclidean distance, cosine similarity, and the overall average frequency of the WEF-Matrix among clients. For benign clients and freeloaders, the server aggregates and distributes different global models only based on their constantly changing frequency differences. In this way, freeloaders obtain a global model without the contribution of model weights from benign clients, preventing them from stealing high-quality models trained during training.

Chapter 14 introduces a method for protecting model copyright in FL using model fingerprints, which belongs to the category of auxiliary detector defense methods. This method relies on honest FL servers to generate fingerprints for models trained in FL and uses these fingerprints and their outputs to train a separate model. Subsequently, a new model called a detector is trained using the feature distribution of key samples to predict the true class encoding of those key samples.

Chapter 15 presents a data modification defense method in the field of federated learning specifically designed to address privacy inference attacks. This method proposes a fake label generator to obtain uniformly distributed labels and then designs a noise generator to protect sensitive attributes. Experimental results show that this method achieves state-of-the-art performance while balancing privacy protection and task accuracy.

Chapter 16 introduces an FL backdoor defense method that belongs to the category of model modification defense methods. This method collects a small dataset of clean samples used for the main FL task training from the server and uses it to generate adversarial samples. By observing the behavior of updated models under adversarial samples, clustering algorithms are employed to select benign models and eliminate others without affecting the main FL task performance. Through extensive evaluation on four datasets and corresponding DNNs, compared with five baselines, this method demonstrates state-of-the-art (SOTA)-defensive performance.

Chapter 9

Detecting Adversarial Examples via Local Gradient Checking



9.1 Introduction

Thanks to the outstanding performance and stable capability, deep neural networks (DNNs) are now widely used in various areas, e.g., image classification [1], object detection [2], and behavior classification [3]. However, DNNs may expose incorrect predictions due to subtle changes of the input, especially on adversarial perturbations. Their undesirable vulnerabilities of security and robustness problems have become a major security concern. These issues will pose grave threat, especially when DNNs are deployed in the safety-critical domains like autopilot [4] and identity recognition [5].

Numerous adversarial attack methods have been proposed from various perspectives, fooling DNNs with different perturbations. They can be classified into white-box attacks and black-box attacks, according to the knowledge of the targeted model. Typical white-box attacks mainly use gradient information to determine misleading perturbations [6–10]. The structure and parameters of the targeted model should be given in advance before implementation of white-box attacks. On the contrary, black-box attacks mislead the targeted model only with the output of it. They are mainly decision based and score based [11–13]. Compared with white-box attacks, perturbations of black-box attacks usually have larger size.

To counter these attacks, far-intensive efforts have been made to identify security-compromised input sources, following two design goals: (1) high detection rate; (2) low computation cost and time complexity. Existing detection methods are brought up mainly based on the feature difference of adversarial examples and benign examples, or based on the model behavior difference activated by them. They computed distance metrics of each layer in the DNN and characterized key properties of the adversarial subspace. However, to achieve good performance against general attacks, they require a large amount of different adversarial examples and long time for feature extraction, which increases the data dependency and computation cost.

When the model is fed with adversarial examples and benign ones, it's more straightforward and general to address the problem of adversary detection from the

aspect of the model’s different behaviors, i.e., neuron activations. As model-based method, Ma et al. [14] extracted value invariants and the provenance invariants from benign training data, and then used them for adversarial detection. The proposed method NIC traverses each neuron in each layer, so it suffers from high computation burden.

Recall that the decision results of DNNs are determined by the aggregation of each channel, benign and adversarial examples activate different values of channels [14, 15]. So we try to find the difference in a certain layer in the DNN through feeding benign and their corresponding adversarial examples. Since activation values in a certain layer are often normalized to $[0, 1]$, they don’t show significant difference before and after attacks. To magnify the difference, we try to take the partial derivative of the output of the layer and propose the concept of *local gradient*, which quantitatively measures the extent of input’s contribution to layer’s activation values. Specifically, adversarial examples generated by different algorithms, as well as noisy examples, show significantly large bound value of the local gradient, i.e., absolute values up to a thousand times of benign ones. This illustrates that large difference in local gradient between benign and adversarial examples can be used to characterize the properties of adversarial examples.

Inspired by this observation, we leverage the local gradient for adversarial detection and further design a general method, against general adversarial examples. First, we calculate the local gradient in a certain layer in the DNN by a few batches of benign and perturbed examples. To generate these perturbed examples, random noises are added on benign examples until they are misclassified. Thus, their local gradient is used to simulate that of the adversarial examples. Then an adversarial detector will be trained to learn the distributions of the local gradient. The well-trained detector can precisely characterize the properties of the input example. Based on general pattern of adversarial examples in the local gradient, our method does not require explicit prior knowledge of specific attacks. It achieves attack-agnostic detection and can be served as an effective tool against general adversarial examples.

Our main contributions are summarized as follows:

- We propose the concept of local gradient, to measure the extent of input’s contribution to layer’s activation values. We observe that adversarial examples show a quite larger bound of local gradient than that of benign ones.
- Based on the observation, we design a detection method for differentiating benign and adversarial examples. Only relying on a few benign examples, our method can precisely detect general adversarial examples, and even misclassified natural inputs.
- Extensive experiments on benchmark datasets have been conducted to verify its performance. Results have shown that our method is superior to the state-of-the-art (SOTA) detection methods against general adversarial examples.

9.2 Related Work

In this section, we review the related work and briefly introduce adversarial attacks and detection methods against them.

9.2.1 Adversarial Attacks

According to whether the attacker requires prior knowledge of the targeted model, adversarial attacks can be divided into white-box attacks and black-box attacks. Based on full knowledge of model structure and parameters, white-box attacks are easy to conduct with small perturbations. Goodfellow et al. [6] proposed FGSM, adding perturbations on the direction where the gradient of the model changes most. Based on it, Kurakin et al. [7] expanded the operation on loss function into several small steps and proposed BIM. Dong et al. [10] introduced momentum term into the iterative process and they proposed momentum-based iterative FGSM (MI-FGSM). Adversarial examples generated by it are more transferable towards models. Similarly, PGD [9] achieves stronger attack by projecting the perturbation to the specified range iteratively. Papernot et al. [8] leveraged critical pixels in the saliency map and designed Jacobian-based saliency map attack (JSMA) method. Besides, Croce et al. [16] combined extensions of PGD attack with other complementary attacks. They proposed AutoAttack, which is parameter free, computationally affordable, and user independent to test adversarial robustness.

Different from white-box attacks, black-box ones fool the model by the model confidence or output labels. Brendel et al. [11] sought smaller perturbations based on the model decision while staying adversarial. PWA attack [12] starts with an adversarial and performs a binary search between the adversarial and the original until the input is misclassified. Similarly, additive uniform noise attack (AUNA) [17] adds uniform noise to the input until the model is fooled. Andriushchenko et al. [18] designed square attack based on random search, which achieves higher success rate in untargeted setting with query efficiency.

9.2.2 Adversarial Detections

Adversarial detectors can distinguish adversarial examples from benign ones, which serves as an alarming bell in the system. Early-proposed detections are mainly based on image pre-processing. They achieve detections by comparing the difference before and after image transformations. Detection results are based on the difference between the original input and the pre-processing input. Liang et al. [19] introduced scalar quantization and smoothing spatial filter for implementing an adaptive noise reduction (ANR) to adversarial examples. Sutanto et al. [20] conducted image recon-

struction by Deep Image Prior network and then compare the outputs of the target model for detection. These methods are easy to implement, but may be compromised when faced with larger adversarial perturbations.

Another family of these methods is based on feature representation of models. Ma et al. [21] characterized dimensional properties of adversarial regions by local intrinsic dimensionality (LID), based on the distance distribution of the example to its neighbors in the feature subspace. Following them, Cohen et al. [22] trained an adversarial detector and proposed NNIF for distinguishing adversarial attacks, which is based on influence function and k-nearest neighbor ranks in the feature space. Yang et al. [23] observed that the feature attribution of adversarial examples near the boundary always differs from those of corresponding benign examples. They further designed ML-LOO through thresholding a scale estimate of feature attribution scores. Besides, Li et al. [24] designed unsupervised adversarial detection on medical imaging systems to screen out adversarial examples, for further facilitating the correct prediction. Such approaches based on feature representations show limitations in computation efficiency, i.e., they require long time for extracting features from multiple layers to train an extra adversarial detector.

9.3 Methodology

Based on the above observation, we leverage the local gradient for detecting adversarial examples. In this part, we describe our method in detail. We further analyze the algorithm complexity of our method and compare it with other methods.

9.3.1 Overview

The workflow of our method is shown in Fig. 9.1. Our method consists of two major components: calculating local gradient and training an adversarial detector AdvD. To begin with, benign and misclassified examples are input to the DNN, respectively. The corresponding local gradient in the chosen layer will be calculated and further used to train the AdvD. Finally, the well-trained AdvD can correctly distinguish benign and misclassified examples.

9.3.2 Calculation of Local Gradient

In this step, we calculate local gradient that will be further used for training the AdvD.

We added random noise towards benign examples until they are misclassified. Then benign examples and their corresponding misclassified examples are fed into

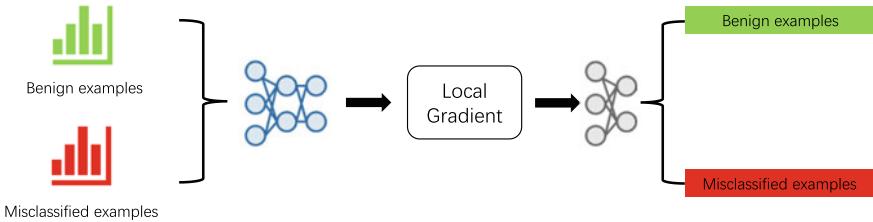


Fig. 9.1 The workflow of our method, including calculating the local gradient and training the detector AdvD

the model for calculating local gradient. The maximum perturbation size of the input examples is limited by 0.25 in $l\text{-}2$ norm. We use random noise-added examples here, instead of adversarial examples of specific attacks. We assume that random noise will be more general to cover various adversarial perturbations. Besides, it is easy to craft with high efficiency.

In this process, in the default setting, the layer between convolutional and dense layer is chosen, e.g., flatten layer or global average pooling. These layers contain both pixel features and high-dimensional features, playing decisive role in classification. The impact of chosen layer will be discussed in the experiment.

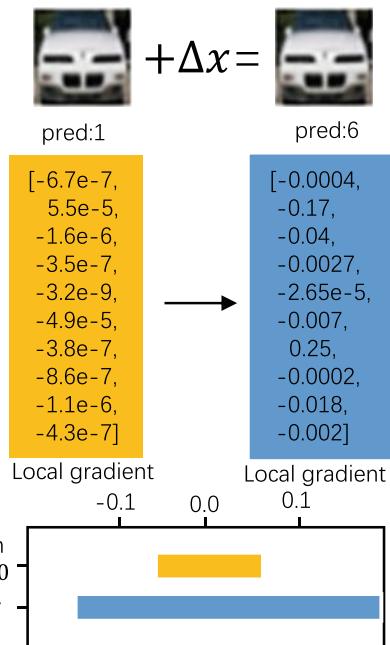
Example. We show how we calculate the local gradient of inputs. We use CIFAR-10 dataset as a running example to illustrate this step, as shown in Fig. 9.2. We train a VGG19 model on CIFAR-10 and use the batch_normalization_19 layer for calculation.

The benign example x classified as class 1 is input to the model. We add noise on it until it is misclassified. The noise-added example is represented as x^* . We then calculate the local gradient of them. Specifically, the absolute value of x^* is significantly larger than that of x . The distinct difference can also be observed from the violin plot. The orange one is much larger than the green one, even the latter is magnified 1000 times.

9.3.3 Detection via AdvD

Local gradient of benign and misclassified examples shows large difference, which we can leverage for detection. Intuitively, one may think a properly set threshold can be a good solution. But different examples may have quite different distributions. As observed from violin plot, although the local gradient bound of adversarial examples is significantly larger than that of benign examples, the specific value is quite different. Besides, the local gradient of benign examples in conv2d_2 in AlexNet locates in the interval of $[-0.01, 0.01]$, consistent with the value of PWA in max_pooling2d_2. So it is hard to determine one threshold for distinguishing benign and misclassified examples among different network architectures in one model, let alone various

Fig. 9.2 Illustration of calculating local gradient on VGG19 of CIFAR-10



models. For a more general detection effect, we design a CNN-based adversarial example detector (AdvD) to determine whether the input data is adversarial.

For each targeted model, we train one detector. AdvD consists of several fully connected layers, whose detailed structures will be stated in the experiment. The input layer size of it is the same shape as the local gradient calculated in the previous step. The local gradient of benign and misclassified examples will be concatenated to generate the training set of AdvD. The input of AdvD is the concatenated local gradient, and the output is the detection result. “benign” means benign example, marked with 0, and “misclassified” means misclassified example, marked with 1. During the training process, the loss function of AdvD is defined as follows:

$$\begin{aligned} loss_{\text{AdvD}} &= CE(\text{AdvD}(F(s)), y), y \in \{0, 1\} \\ F(s) &\in \{\text{Concat}(\mu_i(x), \mu_i(x^*))\}, \end{aligned} \quad (9.1)$$

where $\text{AdvD}(\cdot)$ represents the output of AdvD, x^* is the noise-added misclassified examples, $\text{Concat}(\cdot)$ is concatenate function, and CE is the cross entropy.

The parameters of AdvD are updated by minimizing the loss function. By properly setting training parameters, AdvD can easily detect misclassified examples after training.

9.3.4 Complexity Analysis

Here we analyze the algorithm complexity of our method. Since our method needs to extract the local gradient for detection, the time complexity can be calculated as $T(\text{Our method}) \sim \mathcal{O}(m)$, where m denotes the number of examples to be detected.

As for space complexity, our method needs to restore local gradient for each training example. So the space complexity of it is $S(\text{Our method}) \sim \mathcal{O}(m \times a)$, where a denotes the total number of neurons in the chosen layer.

As for existing detections, LID needs to compute feature distance in each layer, so the running time is magnified by the total number of layers l . So its time complexity is $T(\text{LID}) \sim \mathcal{O}(m \times l)$. On the contrary, ANR only conducts simple transformations towards adversarial examples, the computation complexity is $T(\text{ANR}) \sim \mathcal{O}(m)$. NNIF transverses all examples in the training dataset, so it requires long time for computation: $T(\text{NNIF}) \sim \mathcal{O}(m \times t \times l)$, where t denotes the number of examples in the training dataset. NIC has to traverse all neurons inside the model, time complexity of it is $T(\text{NIC}) \sim \mathcal{O}(m \times n)$, where n denotes the total number of neurons in the DNN.

9.4 Experiments and Analysis

To demonstrate the performance of our method, extensive experiments have been carried out, including the research questions (RQs):

- **RQ1:** Does our method show superior performance in detecting various adversarial attacks?
- **RQ2:** Can our method effectively detect misclassified natural inputs?

9.4.1 Experiment Setup

Datasets: We evaluate the detection effectiveness of our method on CIFAR-10 [25], GTSRB [26], and ImageNet [27]. CIFAR-10 contains 60,000 32×32 RGB images, belonging to 10 and 100 classes. In our experiment, 50,000 examples are used for training and 10,000 for validation set. GTSRB consists of more than 50,000 48×48 German traffic sign images from 43 classes. We split the original training dataset into two parts, i.e., 70% for training set and 30% for validation set. For ImageNet, 11,000 images are selected for training and 2,000 images for testing.

DNNs: We adopt various classifiers on several benchmark datasets. For CIFAR-10 dataset, VGG19 [28] and AlexNet [29] models are adopted. For GTSRB, we train LeNet-5 [30] and ResNet20 [31]. For ImageNet, we train VGG19 and MobileNetV1 [32].



Fig. 9.3 ASR of adversarial examples

Attacks: We use following attacks to generate adversarial examples of different perturbation sizes and distributions: FGSM [6], BIM [7], JSMA [8], PGD [9] as white-box attacks and AUNA [17], PWA [12], Boundary [11] as black-box attacks. Besides, we also conduct AutoAttack (Auto) [16] for evaluation. The implementations of FGSM, BIM, JSMA, PGD, AUNA, PWA, and Boundary are from the Foolbox tool. Attack success rate is shown in Fig. 9.3, respectively

Detection Baselines: The following detection methods are adopted for comparison, including ANR [19], LID [21], NNIF [22], NIC [14], and ML-LOO [23]. They are all recently proposed detection algorithms, representing the SOTA results. We obtained the implementation of ANR, LID, NNIF, and ML-LOO from GitHub. We implement our copy of NIC following the introduced technical approach. As for parameters, we set $k = 10$ and $\text{batch_size} = 200$ for calculating LID. For NNIF, $M = 10$ and C of logistic regression classifier is $1e5$. For ML-LOO, we choose last three fully connected layers for feature extraction. All baselines are configured according to the best performance setting reported in the respective papers.

Evaluation Metrics: The metrics used in the experiments are detailed as follows:

① Classification accuracy: $acc = \frac{N_{true}}{N_{benign}}$, where N_{true} is the number of benign examples correctly classified by the targeted model and N_{benign} denotes the total number of benign examples.

② Perturbation l_2 -norm: $\rho = \|x_{adv} - x\|_2$, where x and x_{adv} are benign and its corresponding adversarial example, respectively, and $\|\cdot\|_2$ represents l_2 -norm.

③ Attack success rate: $ASR = \frac{N_{adv}}{N_{benign}}$, where N_{adv} denotes the number of adversarial examples.

④ Detection rate: $DR = \frac{N_{det}}{N_{mis}}$, where N_{det} denotes the number of misclassified examples detected by detection methods.

⑤ Area Under Curve (AUC) score [33]: The area under receiver operating characteristic curve. The higher, the better.

9.4.2 Detection Against Adversarial Attacks

We focus on the detection results of our method to measure the effectiveness against various adversarial examples.

Implementation Details. (1) 2,000 adversarial examples per attack are generated. Note that these adversarial examples are all generated from the correctly classified images from the training set. (2) We calculate DR against various adversarial attacks for measurement and compare our method with five recently proposed baselines on six models of different datasets. Results are shown in Fig. 9.3, where “Benign” means detection accuracy on benign examples. Besides, the p-value of t-test between our method and each baseline is also calculated, as shown in Fig. 9.4. Two-tail and paired-sample t-test is adopted. p-value larger than 0.05 is bold. (3) We mix 2,000 adversarial examples per attack and 2,000 benign examples, and then calculate AUC score for each detection. Under this setting, VGG19 of CIFAR-10, LeNet-5 of GTSRB, and MobileNetV1 of ImageNet are adopted. Results are shown in Fig. 9.6.

Results and Analysis. In Fig. 9.5, in all cases, our method shows superior detection (almost up to 100%) on various adversarial attacks, when compared with base-

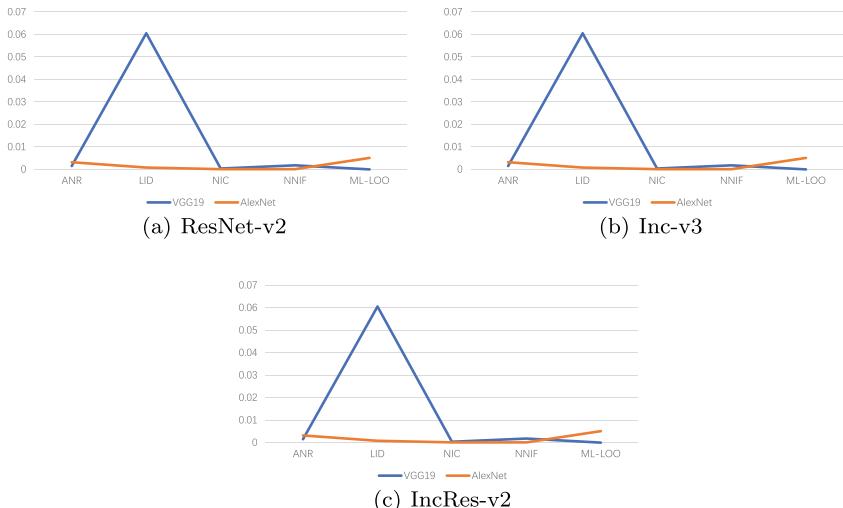


Fig. 9.4 The p-value of t-test on DR between our method and each baseline. Values larger than 0.05 are bold

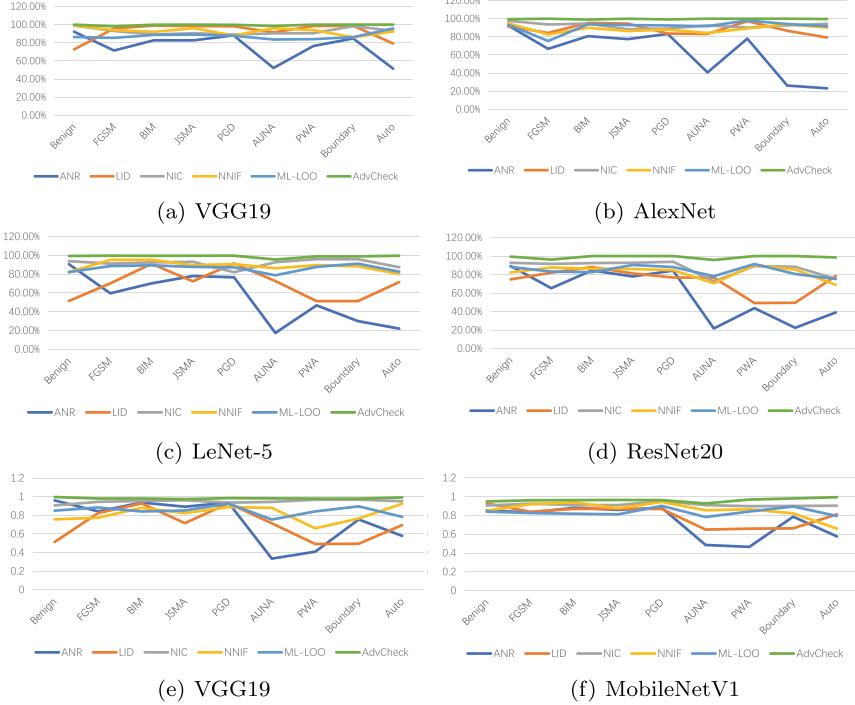


Fig. 9.5 Comparison between AdvCheck and baselines on detecting adversarial examples

lines. And almost all p-values of our method in Fig. 9.4 are smaller than 0.05. Besides, it outperforms both feature-based and model-based detection baselines with a considerable margin, especially on multi-class dataset like GTSRB. Specifically, on LeNet-5 of GTSRB, the DR of our method is 98.82% on average, which is 1.8 times and 1.2 times that of ANR and NNIF. We speculate the reason that our method well captures the large difference between benign and adversarial examples on local gradient, which is easy to distinguish.

Besides, our method shows quite stable detection results on different attacks, regardless of white-box attacks or black-box ones. When dealing with larger perturbations such as AUNA, DR, and AUC don't show large fluctuations. This indicates the generality of our method. By learning the general pattern of attacks in local gradient, various adversarial examples can be detected. On the contrary, as for feature-based ANR, DR decreases when perturbations are larger.

When detecting benign examples, the classification accuracy of our method is the highest. This indicates that our method hardly sacrifices the benign accuracy while completing effective detection. Such large differences, independent on perturbation size and model structures, are easy for our method to distinguish. This also contributes to high AUC in mixture scenario in Fig. 9.6. With regard to baselines, ANR

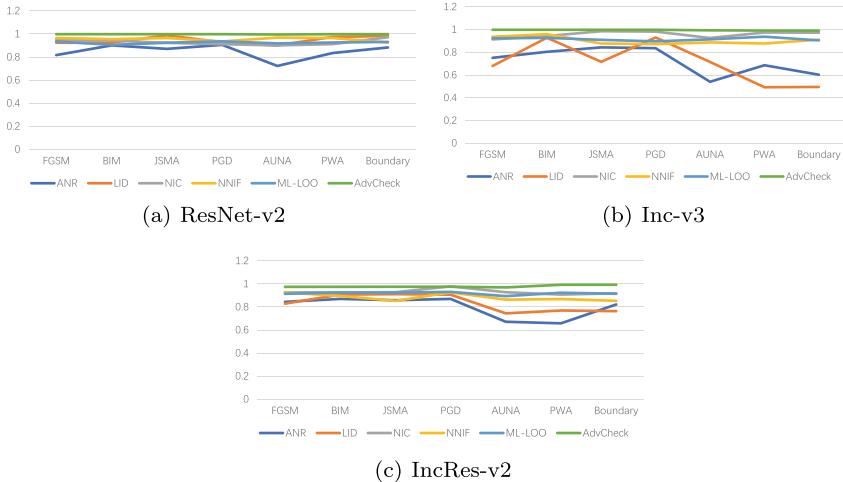


Fig. 9.6 AUC scores between our method and baselines on detecting mixture examples

removes adversarial perturbations together with important pixels, which decreases the accuracy on benign examples.

Answer to RQ1: Our method outperforms the SOTA baselines in two aspects: (1) DR against various attacks ($\sim \times 1.3$ on average); (2) highest DR on benign examples; and (3) higher AUC score ($\sim \times 1.2$ on average) on the mixture of benign and adversarial examples.

9.4.3 Detection Against Misclassified Natural Inputs

We further evaluate our method on two misclassification tasks, including misclassified examples in the training dataset and those due to weather conditions.

Implementation Details. (1) We conduct simple experiments on VGG19 of CIFAR-10. (2) 500 misclassified examples from training dataset (dubbed as “FP”) are selected. For weather setup, 500 misclassified examples are generated by Deep-Xplore [34] to imitate natural weather conditions. (3) The detection performance of our method, measured by DR, will be compared with baselines. The distributions of local gradient of misclassified examples, including adversarial examples (FGSM, PGD, PWA).

Specifically, DR of baselines is around 70% on average against both misclassified inputs while DR of our method is up to 97%. The outstanding performance of our method is mainly because the distribution learned by our method takes into account those misclassified examples, whose local gradient is similar to that activated by adversarial examples (consistent with results shown in Table 9.1). Thus, our method’s detection can be conducted more accurately on those natural examples.

Table 9.1 Detection results

	ANR	LID	NNIF	NIC	ML-LOO	AdvCheck
FP	52.3	65.2	53.4	91.7	70.1	98.3
Weather	58.1	84.2	49.7	93.2	75.3	98.1

Answer to RQ2: Apart from detecting adversarial examples, our method can be applied to characterize misclassified natural inputs, e.g., FP and weather examples. It performs better on DR ($\sim \times 1.4$ on average) than baselines.

9.5 Conclusion

In this chapter, we propose the concept of local gradient, and observe that benign and adversarial examples show quite large difference in the same layer in the DNN. Based on it, we design our method, a general framework for detecting adversarial examples and even misclassified natural inputs. It only relies on a few benign examples to achieve attack-agnostic detection. Extensive experiments have verified that compared with SOTA baselines.

References

1. Zhang, G., Wang, B., Wei, F., Shi, K., Wang, Y., Sui, X., Zhu, M.: Source camera identification for re-compressed images: a model perspective based on tri-transfer learning. *Comput. Secur.* **100**, 102076 (2021)
2. Zhang, H., Ma, X.: Misleading attention and classification: an adversarial attack to fool object detection models in the real world. *Comput. Secur.* **122**, 102876 (2022)
3. Chen, A., Fu, Y., Zheng, X., Lu, G.: An efficient network behavior anomaly detection using a hybrid dbn-lstm network. *Comput. Secur.* **114**, 102600 (2022)
4. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1625–1634 (2018)
5. Rozsa, A., Günther, M., Rudd, E.M., Boult, T.E.: Facial attributes: Accuracy and adversarial robustness. *Pattern Recogn. Lett.* **124**, 100–108 (2019)
6. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, pp. 1–11 (2015)
7. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, pp. 1–14. OpenReview.net (2017)
8. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)

9. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–28. OpenReview.net (2018)
10. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: Proceedings of the IEEE Conference On Computer Vision and Pattern Recognition, pp. 9185–9193 (2018)
11. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–12. OpenReview.net (2018)
12. Schott, L., Rauber, J., Bethge, M., Brendel, W.: Towards the first adversarially robust neural network model on MNIST. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019, pp. 1–16. OpenReview.net (2019)
13. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **23**(5), 828–841 (2019)
14. Ma, S., Liu, Y.: Nic: Detecting adversarial samples with neural network invariant checking. In: Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019), pp. 1–15 (2019)
15. Bai, Y., Zeng, Y., Jiang, Y., Xia, S.T., Ma, X., Wang, Y.: Improving adversarial robustness via channel-wise activation suppressing. In: International Conference on Learning Representations, pp. 1–19 (2020)
16. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: International Conference on Machine Learning, pp. 2206–2216. PMLR (2020)
17. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Additive uniform noise attack in foolbox tool. <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks/decision.html#foolbox.attacks.AdditiveUniformNoiseAttack>
18. Andriushchenko, M., Croce, F., Flammarion, N., Hein, M.: Square attack: a query-efficient black-box adversarial attack via random search. In: European Conference on Computer Vision, pp. 484–501. Springer (2020)
19. Liang, B., Li, H., Su, M., Li, X., Shi, W., Wang, X.: Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Trans. Dependable Secur. Comput.* **18**(1), 72–85 (2021)
20. Sutanto, R.E., Lee, S.: Real-time adversarial attack detection with deep image prior initialized as a high-level representation based blurring network. *Electronics* **10**(1), 52 (2020)
21. Ma, X., Li, B., Wang, Y., Erfani, S.M., Wijewickrema, S.N.R., Schoenebeck, G., Song, D., Houle, M.E., Bailey, J.: Characterizing adversarial subspaces using local intrinsic dimensionality. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings, pp. 1–15. OpenReview.net (2018)
22. Cohen, G., Sapiro, G., Giryes, R.: Detecting adversarial samples using influence functions and nearest neighbors. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020, pp. 14441–14450. Computer Vision Foundation/IEEE (2020)
23. Yang, P., Chen, J., Hsieh, C., Wang, J., Jordan, M.I.: ML-LOO: detecting adversarial examples with feature attribution. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7–12, 2020, pp. 6639–6647. AAAI Press (2020)
24. Li, X., Pan, D., Zhu, D.: Defending against adversarial attacks on medical imaging ai system, classification or detection? In: 2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI), pp. 1677–1681. IEEE (2021)
25. Krizhevsky, A.: Learning multiple layers of features from tiny images. *Handbook of Systemic Autoimmune Diseases* (2009)

26. Stallkamp, J., Schlipsing, M., Salmen, J., Igel, C.: The German traffic sign recognition benchmark: a multi-class classification competition. In: IEEE International Joint Conference on Neural Networks, pp. 1453–1460 (2011)
27. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision* **115**(3), 211–252 (2015)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–14 (2015)
29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3–6, 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114 (2012)
30. LeCun, Y., et al.: Lenet-5, convolutional neural networks **20**(5), 1–14 (2015). <http://yannlecun.com/exdb/lenet>
31. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
32. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: efficient convolutional neural networks for mobile vision applications, pp. 1–9 (2017). [arXiv:1704.04861](https://arxiv.org/abs/1704.04861)
33. Fawcett, T.: An introduction to roc analysis. *Pattern Recognit. Lett.* **27**(8), 861–874 (2006)
34. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: proceedings of the 26th Symposium on Operating Systems Principles, pp. 1–18 (2017)

Chapter 10

A Novel Adversarial Defense by Refocusing on Critical Areas and Strengthening Object Contours



10.1 Introduction

Recently, defenses against adversarial attacks have been studied. Most defenses have evolved along three main directions [1]: training/input modification [2, 3]; model modification [4, 5], for example, to change network parameters or structure; and Network add-ons [6], for example, employing additional networks or applying additional processing to filter the input. Resizing [7, 8] proves to be an efficient defense strategy that eliminates most of the perturbations by changing the size of the pixel matrix. However, the parameters of resizing depend on the dataset and the model and have a significant impact on the defense performance.

It is commonly accepted that adversarial attacks are crafted by adding perturbations to benign images to weakening the feature pixels of the image, thus spoofing the DNN. Several works have tried to account for how well-designed adversarial perturbations can have effects using decision boundaries [9, 10] or distillation models [11, 12]. Different from the present work, we use visualization to focus on the differences between benign and antagonistic instances and further amplify the differences between them, as shown in Fig. 10.1. From left to right, images in the first row are benign, adversarial, and reconstructed examples, where the benign example is categorized by the Inception v3 model [13] as a “hussar monkey”. The adversarial example generated by DeepFool [14] is mislabeled as “baboon”. The second row of images is a visualized heatmap opposite to the first row. In contrast to the visualized heatmap of the benign example, the region of the adversarial example becomes diffuse and the outline of the object becomes blurred, which is one of the reasons for the misclassification. Therefore, we propose a direct and effective defense method that focuses the model’s attention on key feature regions and enhances the object contours.

Motivated by the attention-based DNN interpretation method, we propose a novel defense method, namely, “Refocusing Critical Regions and Strengthening Object Contours”. We employ an attention mechanism to target pixels with adversarial perturbations, eliminate the perturbations, and strengthen the categorized features.

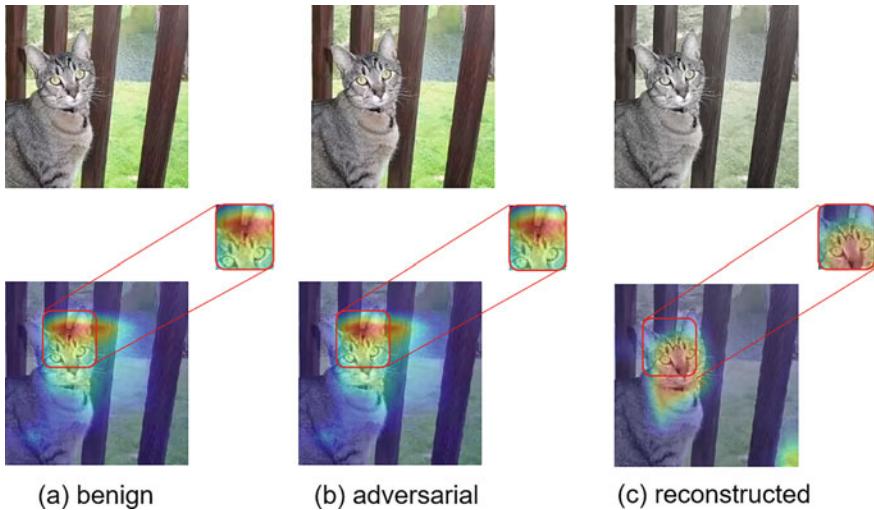


Fig. 10.1 The images in the top row are benign, adversarial, and reconstruction examples in order from left to right. The images in the next row are the attentional visualizations of the corresponding examples in the previous row. From red to blue, the model assigns decreasing weights. Compared to the benign image heatmap, the yellow and red regions in the adversarial image heatmap increase and the object contours become blurred, leading to misclassification. Whereas, in our method reconstruction example, the critical regions are greatly reduced and the object contours are enhanced. Therefore, the reconstructed image is correctly recognized. **a** label = (371, hussar monkey), confidence = 0.918, logits = 10.5, **b** label = (372, baboon), confidence = 0.462, logits = 8.8, **c** label = (371, hussar monkey), confidence = 0.682, logits = 12.1

Since the perturbations added in the critical regions are weakened, we believe that it is effective in reconstructing adversarial examples and thus defends against attacks. As shown in Fig. 10.1c, the reconstructed examples are correctly labeled after our method processing. Observing the heatmap of the reconstructed image, the focus region is more concentrated and the object contours are clearer than the adversarial image. The presented our method is an easy-to-implement direct defense strategy.

Our work's main contributions are summarized as follows:

- (1) We performed heatmap comparisons between clean and adversarial examples in terms of attention weights, and determined that model misclassification was due to adversarial perturbations that deviated from key features of the model. We leverage the direct and effective defense strategy of refocusing on key object contours.
- (2) We presented a novel adversarial paradigm reconstruction method, in particular, our method. Our method is based on pixel-attention weights, and intended to focus the model's attention on the key feature regions and reinforce the object contours of the input image. It achieved quite effective defense. Moreover, current state-of-the-art defense methods can be greatly improved even when cascaded with our method.

- (3) In order to perform a comprehensive analysis of our method, we conducted a number of experiments, including defense success rates against white-box and black-box attacks, pre- and post-defense obfuscation matrices, heatmap visualization comparisons, and contour blur visualization comparisons.

10.2 Related Works

10.2.1 Defense for DNNs

With the huge threat posed by adversarial attacks, defense strategies are increasingly being researched [1]. Defenses have evolved along three main directions: training/input modification, network modification, and network attachment.

10.2.1.1 Training/Input Modification

Goodfellow et al. [2] and Huang et al. [3] exploited adversarial training to enhance the robustness of DNNs against adversarial attacks by injecting adversarial examples into the training set. Xie et al. [7], Dziugaite et al. [15], and Guo et al. [8] discovered that the intensity of the attack can be reduced by introducing random resizing, padding, and JPEG compression into the image. Wu et al. [16] designed an attention-based defense framework that corrects the predicted attentional mapping and keeps the attentional region between the adversarial image and the clean image.

10.2.1.2 Network Modification

Papernot et al. [4] exploited defensive distillation to recover its robustness to small-scale adversarial perturbations using network knowledge. Dhillon et al. [5] presented stochastic activation pruning for robust adversarial defense, where defenses are gained by pruning a random subset which activates the layer function and expanding the survivors.

10.2.1.3 Network Add-On

Akhtar et al. [6] presented a novel perturbation rectifier network (PRN) to add to the primal network. Without tuning any parameter in the primal network, the network gains the ability to defend against pervasive adversarial perturbations through the add-on. Lee et al. [17] used a generative adversarial network to enhance the defense capability of the DNN.

10.2.2 Attention Mechanism for DNNs

Visual attention mechanisms have been proven valid in a variety of structural prediction tests such as image [18] and video [19] captioning, scanpath prediction [20], and visual question answering [21]. Informed by the biological visual system, the attentional mechanism makes the final decision by focusing attention on a few important parts rather than the entire visual space. Chen et al. [22] developed a novel convolutional neural network structure, SCA-CNN, by combining the channel attention mechanism with the spatial attention mechanism to enhance the feature extraction capability. Different from the attention mechanism used for feature extraction and attack in SCA-CNN, we propose pixel channel attention and pixel plane attention in our method to filter adversarial perturbations, which refocuses the key regions and strengthens the object contours.

Hidden layer feature visualization is an important instrument for demonstrating key representations for deep model learning. Heatmaps [23] are a useful and successful visualization tool to visualize the weights assigned to an image by the model, since the human visual system cannot see adversarial perturbations added to a clean image. The use of heatmaps makes it easy to compare the differences between adversarial and benign examples, as shown in Fig. 10.1.

10.3 Methodology

Our method aims to enhance the defense of DNNs against adversarial attacks through pixel-attention mechanism and feature maps. Its major framework is presented in Fig. 10.2. First, adversarial examples are created from different attack methods, i.e., FGSM and MI-FGSM. After that, we input the adversarial examples into the target DNN. The examples are primed with $H \times W \times 3$, where H and W are the height and width of the input image. Then, a feature map based on the shallow layer of the target DNN is created to extract features. Our method employs two attention mechanisms: pixel channel attention and pixel plane attention. The former is used to find critical regions of the model and the latter is used to enhance the contours of the object. Finally, the attention weights are applied by multiplying them with the adversarial paradigm to rebuild the image. The pixel channel attention and pixel plane attention mechanisms are differentiated and specific steps are elaborated according to the order in which they are implemented.

10.3.1 Feature Map Extraction

When dealing with adversarial examples, the first task is to identify the feature pixels that determine the image labels. Convolutional kernels are employed as filters to extend the features of the image. Different convolution kernels can be used to

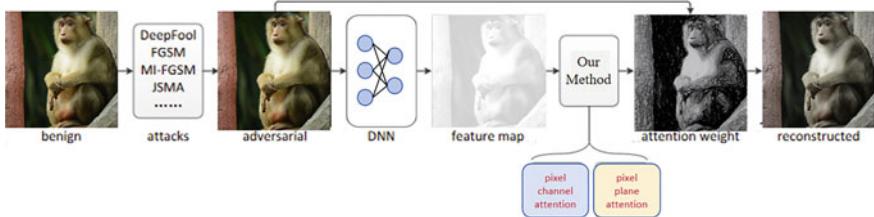


Fig. 10.2 Framework of Our method. The counter-examples are first used in the shallow layer of the target DNN to generate a feature map. After that, attention weights can be obtained from pixel channel attention and pixel plane attention

fetch information about different features in the same image. Through stacking these features, the model can learn the image features of the visual abstraction layer.

Features derived from deeper layers are broader and more abstract than those extracted from shallower layers. However, it is often the case that deeper layers amplify features against perturbations, which may not capture the actual features of a clean example. Thus, output from shallower layers should be used when mapping features to capture more spatial information about contours and feature regions. Here, we use the DNN first layer output as the feature map. The output feature map size is denoted as $H' \times W' \times C$. H' and W' , different from H and W , denote the output height and weight of the feature layer. C denotes the depth of the channel of the feature layer. The different channels of the feature maps are stacked and then the shallow network features are rebuilt to the same dimensional space as the input image by bilinear interpolation. In other words, we resample the feature maps up to $H \times W \times C$. The upsampled feature map is tagged as $I_{fm} \in \mathbb{R}^{H \times W \times C}$. Feature map extraction helps to learn the image's spatial and contour information.

10.3.2 Pixel Channel Attention

Reconfiguration defense aims to eliminate adversarial perturbations in our method. Thus, we employ pixel channel attention to focus on key feature regions. Attention mechanisms are divided into two categories: hard attention and soft attention. The former is employed to select regions that are most likely to be attended to, and the latter is utilized to average the spatial features and the attention weights. To filter adversarial perturbations, we adopt pixel channel attention to attend to the image with hard attention.

Specifically, we first compute the pixel channel attention weights. The weights assigned to different pixels by the pixel channel attention mechanism are defined as pixel channel attention weights, denoted as W_c . The I_{fm} and the adversarial example matrix $I_{adv} \in \mathbb{R}^{H \times W \times 3}$ are separately translated by swapping the third dimension with the first and second dimension. Thereafter, the former is reshaped to $V_{fm} \in$

$\mathbb{R}^{C \times S}$ and the latter to $V_{adv} \in \mathbb{R}^{3 \times S}$, where $S = H \times W$. The similarity between V_{fm} and V_{adv} is calculated as

$$\alpha = V_{adv} \times V_{fm}^T, \quad (10.1)$$

where “ \times ” denotes matrix multiplication. Linear normalization applied to compute the pixel channel attention weights as

$$W_c = \frac{\alpha - \min(\alpha)}{\max(\alpha) - \min(\alpha)}, \quad (10.2)$$

where $\max()$ represents the maximum value in the similarity matrix and $\min()$ denotes the minimum value. Normalization maps the resemblance to an desired interval, and $W_c \in \mathbb{R}^{3 \times C}$ denotes the pixel channel attention weights.

A single 2D convolutional network is used to generate a channel attention image based on the feature map and the pixel channel attention weights based on the convolutional filter. The vector labeling of the channel attention image is $V_c \in \mathbb{R}^{H \times W \times C}$.

10.3.3 Pixel Plane Attention

Pixel plane attention aims to pay attention to different pixels depending on their feature weights. Pixels with higher weights often receive more attention. Rather than treating every pixel equally, the pixel plane attention mechanism pays more attention to feature pixels.

We wish to highlight the key pixels in the image and weaken the antagonistic perturbation pixels. The pixels with large differences can be easily identified by calculating the similarity between V_c and the antagonistic examples. Based on the similarity, we can determine the attributes of the pixel, whether it is a perturbed pixel, a feature pixel or a normal pixel. The weights in this step are called pixel plane attention weights, denoted as W_p . In this step, we need to enhance the features further on the basis of W_p to make the object outline clearer.

The formula for calculating the similarity distance between the channel attention image V_c and the adversarial examples I_{adv} is calculated as

$$\beta = \text{ave}(I_{adv} * V_c), \quad (10.3)$$

where “ $*$ ” denotes the matrix multiplication of the corresponding elements, and $\text{ave}(\cdot)$ represents the average function in the channel. Afterwards, we compute the pixel plane attention weight as $W_p \in \mathbb{R}^{H \times W \times 1}$ according to Eq. (10.2).

In order to enhance the features further and differentiate the object outlines, we adopt a traversal method W_p and set a threshold value $/gamma$ as follows:

$$W_p(i, j) = \begin{cases} 0, & \text{if } W_p(i, j) < \gamma, \\ 1, & \text{if } W_p(i, j) > \gamma. \end{cases} \quad (10.4)$$

Thus, the feature pixels are augmented and the object contours are sharpened. The corrected matrix is a pixel plane attention map, which can be represented as a grayscale image, recorded as $I_{map} \in \mathbb{R}^{H \times W \times 1}$.

In each channel, the adversarial image is rebuilt by multiplying I_{map} with the corresponding elements of the adversarial example matrix, as follows:

$$I_{rec} = I_{adv} * I_{map}, \quad (10.5)$$

where $I_{rec} \in \mathbb{R}^{H \times W \times 3}$. The effect of γ on experiment results will be detailed later.

10.4 Experiments

To validate the performance of our method, we conducted comprehensive experiments. Compared with the baseline, our method, especially the defense strategies cascaded with our method, achieves best-in-class performance against a variety of white-box and black-box attacks. We also visualize the heatmap, confusion matrix, and profile ambiguity of the defense methods. We also analyze the parameter sensitivities, i.e., threshold γ and feature layers. Finally, we provide a comparison of the time complexity of our method with other defense methods.

10.4.1 Setup

Datasets: We used simple datasets such as MNIST¹ Fashion-MNIST (FMNIST)² CIFAR-10³ and ImageNet⁴ to testify defensibility. MNIST consists of 70,000 grayscale images of handwritten numbers and 28×28 pixel digital images from 0 to 9. Fashion-MNIST contains 60,000 training examples and 10,000 test examples. Each example is a 28×28 grayscale image linked to a label in one of the 10 categories. The CIFAR-10 dataset comprises 60,000 32×32 color images in 10 categories with 6,000 images in each category. 50,000 images were utilized for training and 10,000 images were utilized for testing. ImageNet is a project for the recognition of computer vision systems and is one of the most extensively used databases. It includes more than 2 million images in 1,000 categories. We selected 5,000 images for our experiment.

¹ <http://yann.lecun.com/exdb/mnist>.

² <https://github.com/zalandoresearch/fashion-mnist>.

³ <http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>.

⁴ <http://www.image-net.org>.

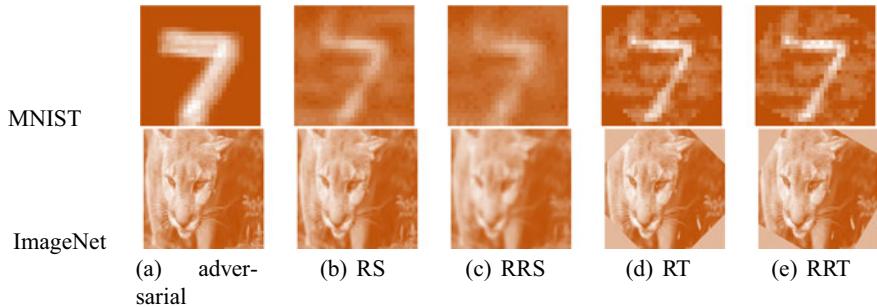


Fig. 10.3 MNIST and ImageNet are used as examples to visualize the input modifications for different datasets

DNNs: In order to account for the generalizability of the experimental results, we used several deep models with different convolutional structures. For MNIST, we elaborated two self-training models, CNN-A and CNN-B. For CNN-A and CNN-B, the batch size was set to 32, and the learning rate was set to 0.001. For FMNIST, we trained two models. One is based on AlexNet [24], and the other, named CNN-FM. For CIFAR-10, we trained two models based on ResNeXt [25] and VGG16 [26]. For ImageNet, we adopted Inception v3 (Inc-v3) [13], Inception-resnet-v2 (Inc-Res-v2) [27], and Resnet v2-101 (Res-v2) [28].

Attack methods: Our method and cascade defense methods have been implemented for different attack methods, including DeepFool [14], FGSM [2], MI-FGSM [29], JSMA [30], L-BFGS [31] as white-box attacks, Gaussian blur attack (GBA), contrast reduction attack (CRA), and additive uniform noise attack (AUNA) as black-box attacks. As valid and less computationally expensive attacks, they have different principles to produce perturbations at different scales, thus providing various adversarial examples to demonstrate the effectiveness of the our method and cascade methods. All attacks are realized by Foolbox.⁵

Due to computational speed and hardware load considerations, we randomly generated and selected 1000 adversarial examples with 100% attack success rate for each dataset to be used in all defense experiments.

Defense baseline: For a fair comparison with our method, we used input modifications as a baseline, including resize (RS), random resize (RRS), rotate (RT), and random rotate (RRT), which are easy to operate and widely used defense policies [7]. For MNIST and FMNIST, we have a padding value of 0. For CIFAR-10 and ImageNet, the padding value is 128. The specific example of input A concrete example of an input modification manipulation is shown in Fig. 10.3.

Metrics: Regarding the metrics set in our experiments, defense success rate (DSR) evaluates the defense ability against adversarial attacks, while attack fail rate (AFR) measures the performance of defenses against targeted attacks exclusively. In addition, the accuracy rate indicates the impact of the defense method on benign instances.

⁵ <https://foolbox.readthedocs.io/en/latest>.

$$\text{DSR} = \frac{n|_{l_{\text{rec}}=l_{\text{ben}}}}{N_{\text{adv}}}, \text{AFR} = \frac{n|_{l_{\text{rec}} \neq l_{\text{adv}}}}{N_{\text{adv}}}, \text{accuracy} = \frac{n_{\text{ben}}}{N_{\text{ben}}}, \quad (10.6)$$

where l_{rec} , l_{ben} , and l_{adv} denote the labels of reconstructive, benign, and antagonistic examples, respectively. N_{ben} and N_{adv} represent the total number of benign and antagonistic examples, respectively. n_{ben} is the amount of reconstructed benign examples correctly identified. The DSR and AFR of the no-defense training model are both 0%.

Perturbation size $\bar{\rho}_{\text{adv}}$ is defined by

$$\bar{\rho}_{\text{adv}} = \frac{1}{N} \sum_{i=1}^N \|x_{\text{adx}}^i - x_{\text{ben}}^i\|_2, \quad (10.7)$$

where x_{adx}^i and x_{ben}^i denote adversarial examples and benign ones, N denotes the total number of examples, and $\|\cdot\|_2$ denotes the L_2 norm.

In order to clearly show how well the model recognizes different categories after attack and defense, we choose a confusion matrix to show FP (false positive rate) and TP (true positive rate). A confusion matrix was chosen to show the FP (false positive) rate and the TP (true positive) rate. In the case of unbalanced test instances, the accuracy does not provide a fair measure of the model's performance. We used ROC (receiver operating characteristic) curve in our experiments to better validate the robustness of the defense. We set the threshold step to 0.01 to obtain the smoothest curve possible. The horizontal axis of the ROC curve represents the FPR (false positive rate), and the vertical axis represents the TPR (true positive rate). The ROC curve can only qualitatively indicate the model and the defense method's performance. Therefore, the AUC (area under curve) is used as a quantitative measure of performance. The AUC value is equivalent to the area created by the ROC curve and the FPR axis, and ranges from (0.5, 1).

We hypothesize that successful attacks on adversarial examples are associated with fuzzy contours, while our method defends the model by removing fuzzy regions. To demonstrate this, we use heatmaps to qualitatively show the degree of blurring of the examples. Furthermore, we quantify the profile information of the I_{map} based on the Laplacian [32] and Canny [33] and define it as the profile ambiguity. When the value of contour blur is large, I_{map} includes more high-frequency information, which influences the focus area that the model attends to. The contour ambiguity computed by the Laplace operator and the Canny operator is defined as CBDL and CBDC, respectively.

$$\begin{aligned} \text{CBDL} &= \text{Laplacian}(I_{\text{map}}) \\ \text{CBDC} &= \text{Canny}(I_{\text{map}}, 50, 150), \end{aligned} \quad (10.8)$$

where $Laplacian(\cdot)$ and $Canny(\cdot, \cdot, \cdot)$ are functions from opencv-python package,⁶ 50 and 150 are the thresholds of Canny operator, I_{map} is obtained by Eq. (10.4).

10.4.2 Defense Against White-Box Attack

Figures 10.4, 10.5, 10.6, and 10.7 show the our method, RRS, RS, RRT, RT, and other cascading methods' DSR, AFR metrics on different datasets, models, and attacks. Only the pixel channel attention mechanism works well on simple datasets such as MNIST and FMNIST. Therefore, we did not conduct experiments on MNIST using the pixel plane attention mechanism.

The defense analysis against white-box attacks can be divided into two parts. The first is the comparison of different defense methods and the second is the comparison of various cascading strategies. From Figs. 10.4, 10.5, 10.6, and 10.7, it can be seen that random operations degrade the defense performance, e.g., on different datasets, the DSR values of RRT and RRS are inferior to those of our method, RT and RS. Furthermore, it can be concluded that input modification operations such as RS and RT can play an effective role in defense only if the parameters are set appropriately. It can be seen that for simple datasets such as MNIST and FMNIST, the DSR values of our method and the AFR values of the target attacks are slightly better or close to RS and RT. On the other hand, on large and complex ImageNet datasets, the DSR and AFR values of RS and RT are better than those of our method.

Generally, a cascade of two defenses significantly enhances the effectiveness of the defense. DSR and AFR have the highest performance when RS (or RT) is cascaded with our method. As the structure of the model deepens from CNN-A to Res-v2, the DSR of different defense methods decreases, while the average DSR of the cascade method (RS/RT+our method) is still close to 80% and AFR value close to 90%. In addition, for various attacks, the our method and RS/RT methods are less capable of weakening the large perturbations generated by FGSM, but cascading the two methods can greatly enhance the defense performance.

10.4.3 Defense Against Black-Box Attack

This section discusses the resilience of defense models against black-box attacks, in which the attacker is unaware of the target model and estimates the gradient of the model based on confidence and decision-making. We employed GBA, CRA, and AUNA from Foolbox for the black-box attack. We selected 1000 random adversarial instances to observe the defense effect of our method. Based on the analysis in Sect. 10.4.2, we learned that random manipulation can negatively affect the defense results. Figures 10.8, 10.9, 10.10, and 10.11 compare the performance of our method, RS, RT and their cascade methods on different datasets and models.

⁶ <https://pypi.org/project/opencv-python/>.

Fig. 10.4 The performance comparison of defense against white-box attack on MNIST datasets

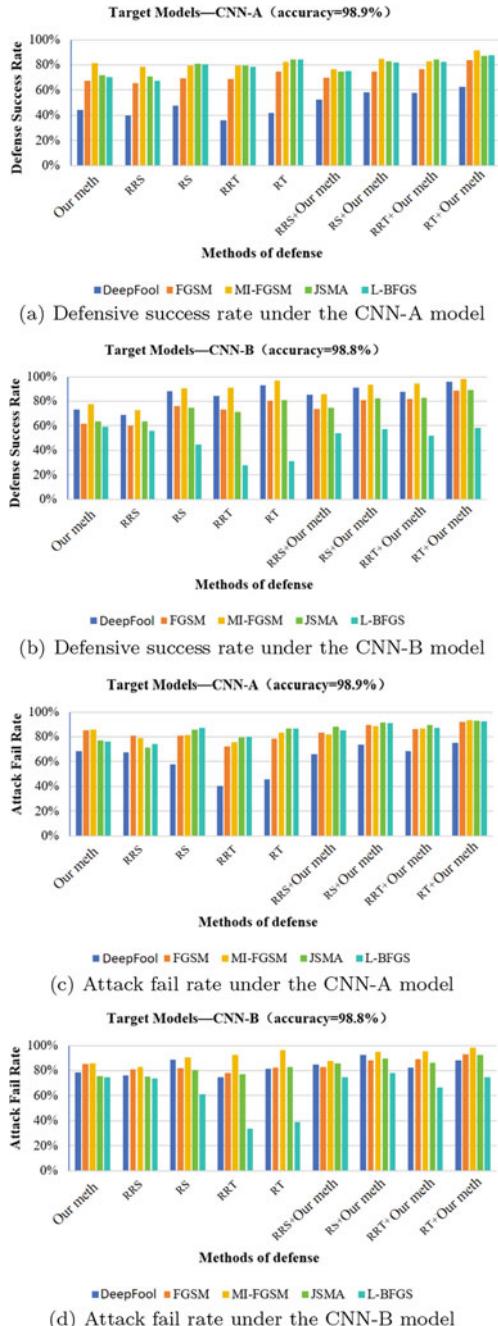


Fig. 10.5 The performance comparison of defense against white-box attack on FMNIST datasets

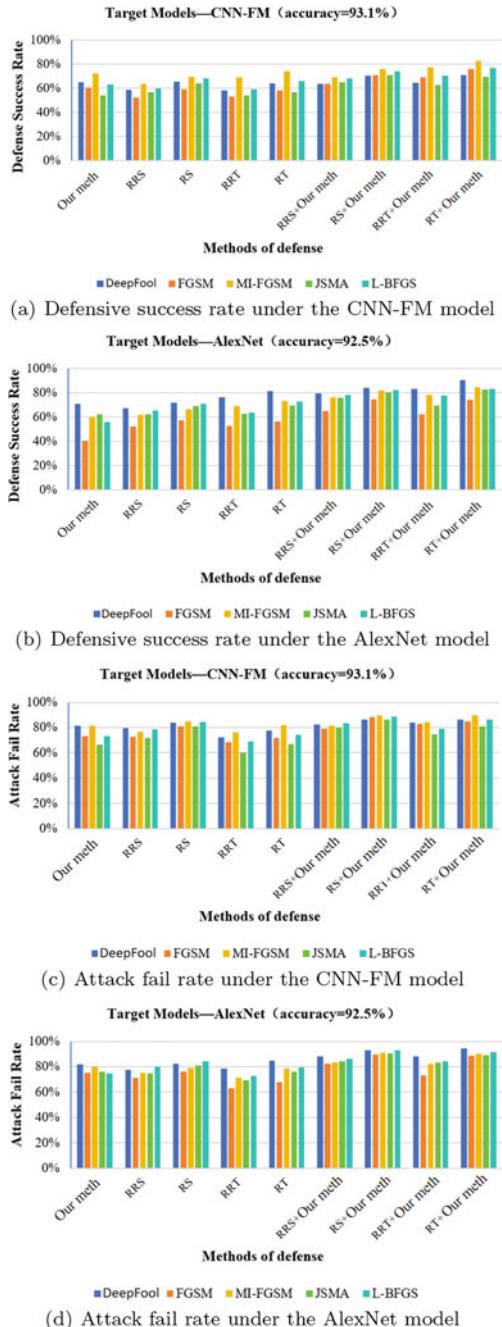


Fig. 10.6 The performance comparison of defense against white-box attack on CIFAR-10 datasets

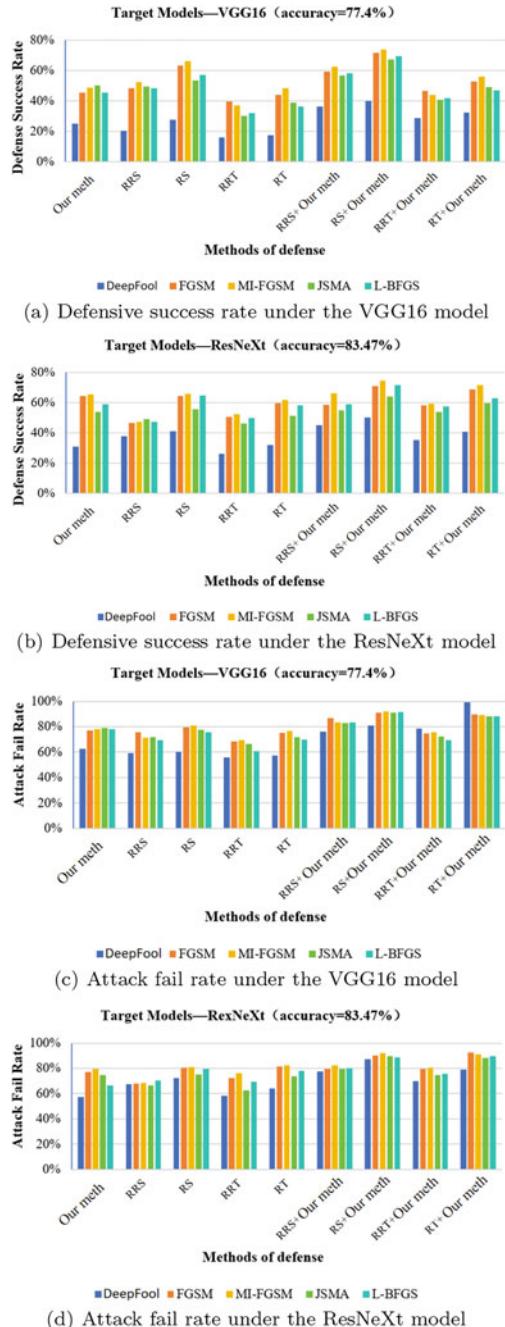
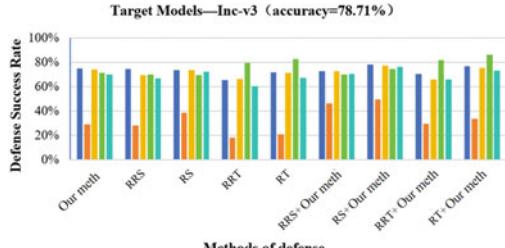
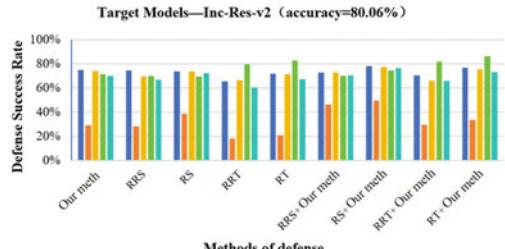


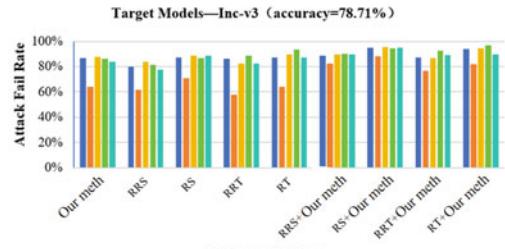
Fig. 10.7 The performance comparison of defense against white-box attack on ImageNet datasets



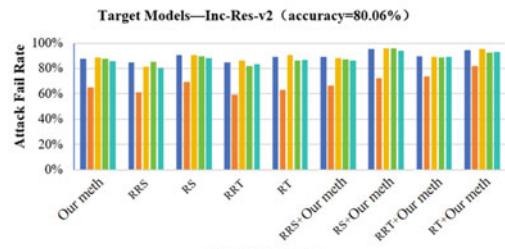
(a) Defensive success rate under the Inc-v3 model



(b) Defensive success rate under the Inc-Res-v2 model

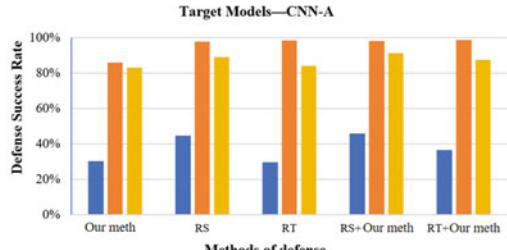


(c) Attack fail rate under the Inc-v3 model

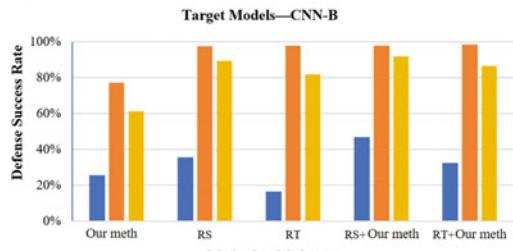


(d) Attack fail rate under the Inc-Res-v2 model

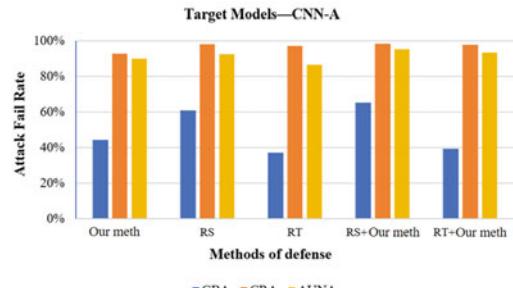
Fig. 10.8 The performance comparison of defense against black-box attack on MNIST datasets



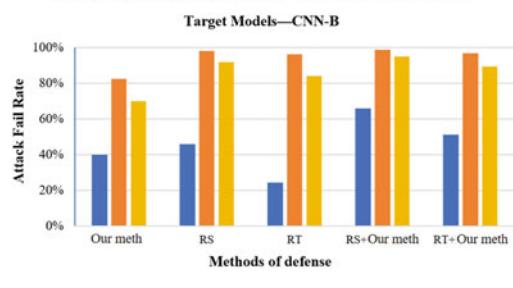
(a) Defensive success rate under the CNN-A model



(b) Defensive success rate under the CNN-B model

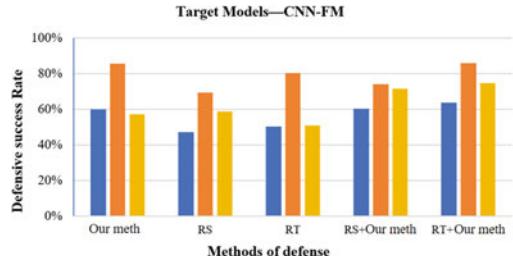


(c) Attack fail rate under the CNN-A model

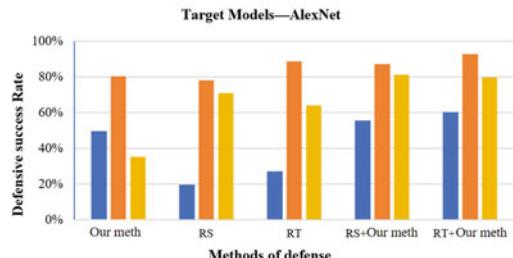


(d) Attack fail rate under the CNN-B model

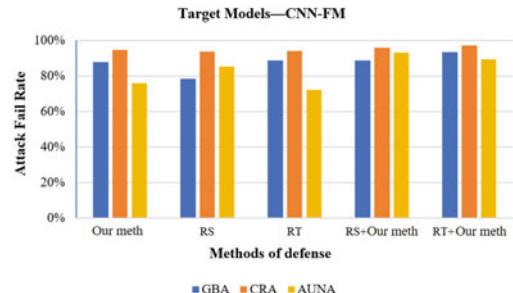
Fig. 10.9 The performance comparison of defense against black-box attack on FMNIST datasets



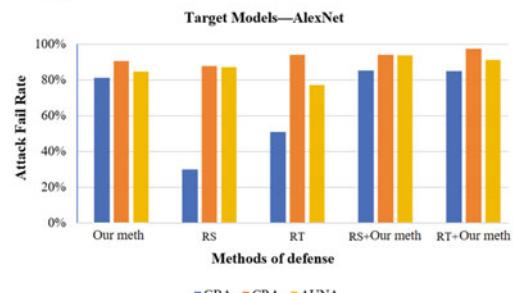
(a) Defensive success rate under the CNN-FM model



(b) Defensive success rate under the AlexNet model

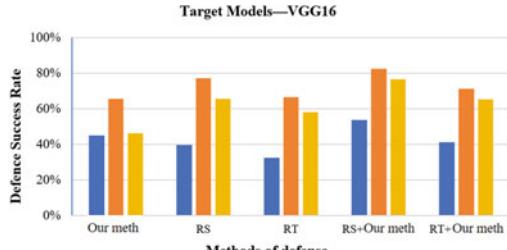


(c) Attack fail rate under the CNN-FM model

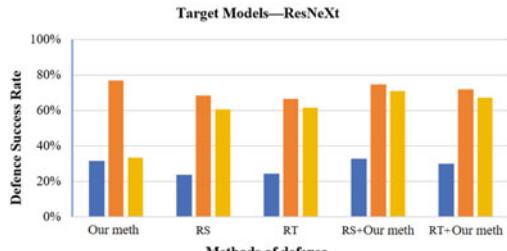


(d) Attack fail rate under the AlexNet model

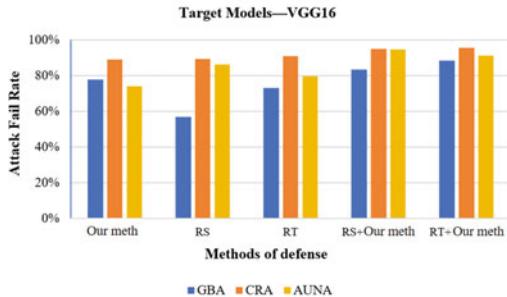
Fig. 10.10 The performance comparison of defense against black-box attack on CIFAR-10 datasets



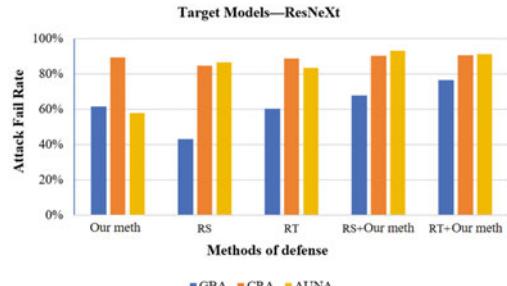
(a) Defensive success rate under the VGG16 model



(b) Defensive success rate under the ResNeXt model

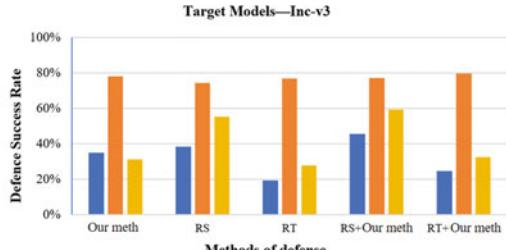


(c) Attack fail rate under the VGG16 model

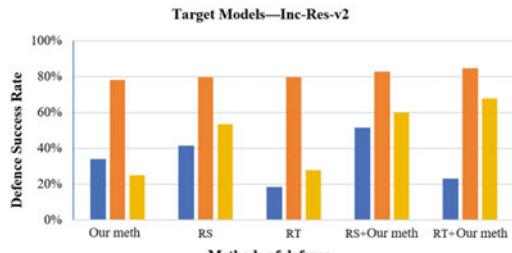


(d) Attack fail rate under the ResNeXt model

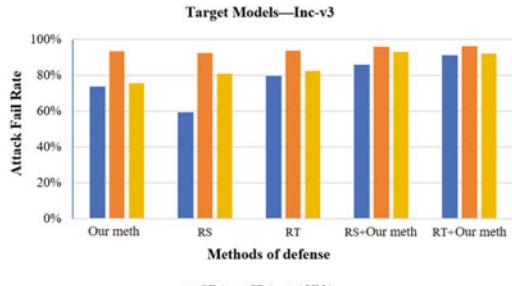
Fig. 10.11 The performance comparison of defense against black-box attack on ImageNet datasets



(a) Defensive success rate under the Inc-v3 model



(b) Defensive success rate under the Inc-Res-v2 model



(c) Attack fail rate under the Inc-v3 model



(d) Attack fail rate under the Inc-Res-v2 model

It was discovered that our method has the capability to defend against both black-box and white-box attacks. Our method has lower DSR values than RS and RT methods in both MNIST and ImageNet datasets, but outperforms them in FMNIST and CIFAR-10 datasets. This indicates that the distribution of different datasets affects the defense performance of our method. The defense capability of the cascade method remains stable even when the complexity of the dataset increases. Compared with RS and RT, the cascade method is more effective in weakening the effect of perturbation on image classification and significantly enhances the ability to change the target labels of adversarial instances. Our method undoubtedly improves the robustness of DNNs against adversarial instances.

10.4.4 Visualization in Cascade Process

10.4.4.1 The Heatmap Visualization

Based on the previous analysis, we know that the performance of RS (or RT) is better than that of RRS (or RRT), and the cascade method is superior to either approach. To intuitively test the effectiveness of our method, we took RS as an example to visualize the heatmap of RS, our method, and RS+our method.

The RS method alters all pixels of the whole image and in most cases succeeds in acting on the perturbed pixels, completing the defense. Our method plays a crucial role when RS fails to weaken and eliminate the perturbation. It utilizes the attention mechanism to put the action region directly on the feature pixel, which is more intentional and ensures the defense effect. Thus, cascading the RS method with the our method can achieve better defense effect. The defense effect remains stable when applying RS+our method to complex datasets and complex models.

Figure 10.12 displays the weights assigned by the network to various images under the attention mechanism. When the defense is successful, the visualization of the post-defense image is closer to the benign image. We can draw the conclusion that these defenses eliminate the adversarial perturbations to some extent, concentrate the critical regions, and enhance the robustness of the model against adversarial attacks. However, even if the defenses fail, the yellow and red regions do not shrink. As a result, the image visualization results are closer to the adversarial image, which can still be misclassified. After cascading RS and our method, our method greatly shrinks the scale of the yellow and red regions and zeroes out irrelevant pixel points in the image, which suggests that the defense effect and object contours are enhanced.

10.4.4.2 The Confusion Matrix Visualization and ROC Curve

In the categorization task, we not only want to ensure that the accuracy is enhanced, but also the precision. In order to visualize the TP and FP of the model after the attack and defense, we show the recognition results through the confusion matrix. Figure 10.13 illustrates the visualization of the confusion matrices of different models

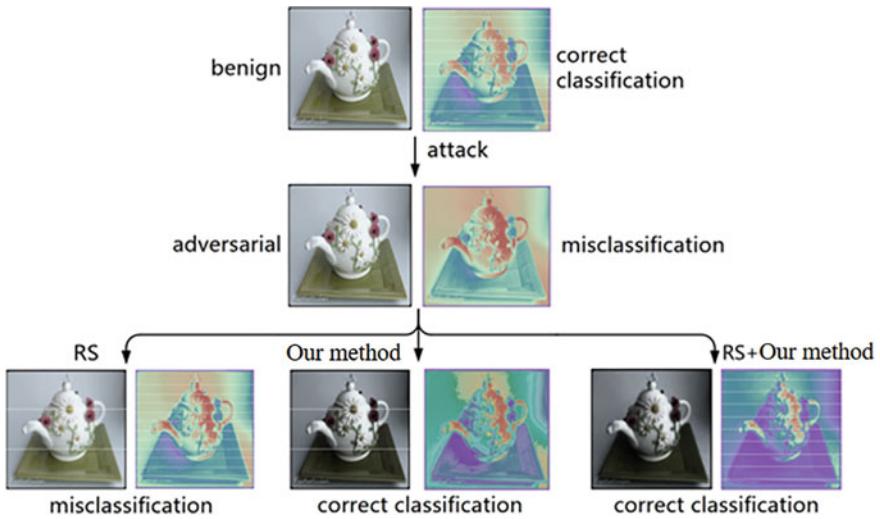


Fig. 10.12 The comparison of visualization heatmaps in different defense processes

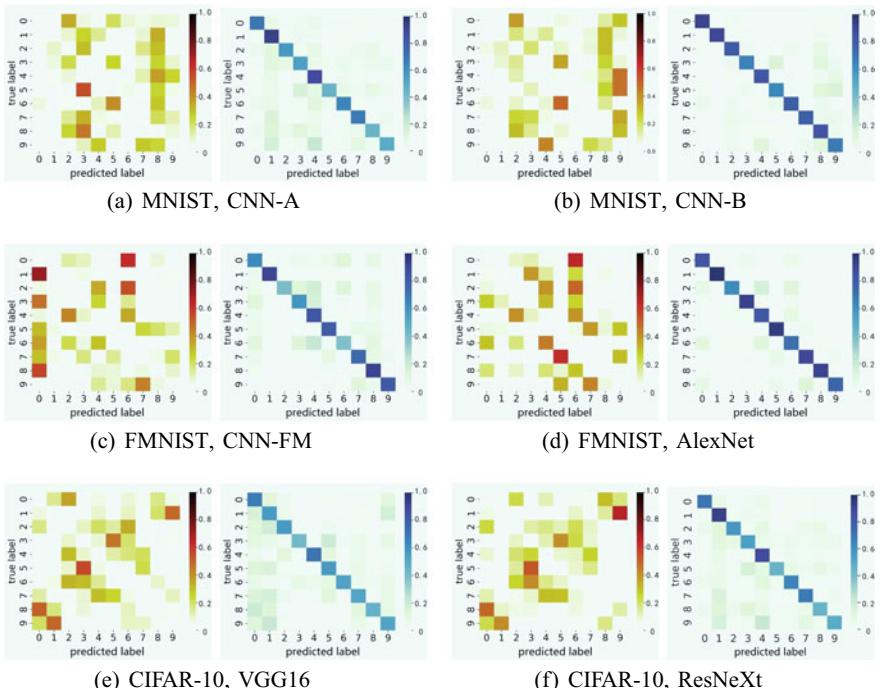


Fig. 10.13 The visualization of confusion matrix of different models on different datasets, where the red and blue bars denote the results of the attack and defense, respectively

on different datasets, where the red and blue bars indicate the results of attack and defense, respectively. Vertical scales are the ground-truth labels and horizontal scales are the labels predicted by the models. The color intensity of the squares on the diagonal from the top left to the bottom right indicates the magnitude of the TP values, and the color intensity of the remaining areas represents the FP values. In the red bars, the confrontation examples are all misidentified, i.e., $TP = 0$. Meanwhile, the color intensity of the squares on the vertical direction can identify the vulnerable class of the model. In the blue bar graph, the blue areas are mainly concentrated on the diagonal from the upper left corner to the lower right corner, which indicates that the adversarial examples are almost correctly identified after defense, with high TP values. In addition, the color of the remaining regions except the diagonal line is almost white, which indicates that the FP value is close to zero.

10.5 Conclusions

In this chapter, we observe the attention weighting relationship between benign, adversarial, and reconstructed examples. Motivated by this finding, we try to focus the DNN’s attention on key feature regions and strengthen the object contours. We present our method, which is an efficient defense algorithm that utilizes feature graph and attention mechanisms to effectively counter adversarial attacks. It can be used to defend against both white-box and black-box attacks without retraining the classifier. The performance of RCA-SOC is quite close to that of RS and RT, and the defense performance can be significantly enhanced by cascading it with other defense methods.

References

1. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey (2018). [arXiv:1801.00553](https://arxiv.org/abs/1801.00553)
2. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples (2014). [arXiv:1412.6572](https://arxiv.org/abs/1412.6572)
3. Huang, R., Xu, B., Schuurmans, D., Szepesvári, C.: Learning with a strong adversary (2015)
4. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 582–597. IEEE (2016)
5. Dhillon, G.S., Azizzadenesheli, K., Lipton, Z.C., Bernstein, J., Kossaifi, J., Khanna, A., Anandkumar, A.: Stochastic activation pruning for robust adversarial defense (2018). [arXiv:1803.01442](https://arxiv.org/abs/1803.01442)
6. Akhtar, N., Liu, J., Mian, A.: Defense against universal adversarial perturbations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3389–3398 (2018)
7. Xie, C., Wang, J., Zhang, Z., Ren, Z., Yuille, A.: Mitigating adversarial effects through randomization (2017). [arXiv:1711.01991](https://arxiv.org/abs/1711.01991) (2017)

8. Guo, C., Rana, M., Cisse, M., van der Maaten, L.: Countering adversarial images using input transformations. arXiv preprint [arXiv:1711.00117](https://arxiv.org/abs/1711.00117)
9. Bhagoji, A.N., He, W., Li, B., Song, D.: Practical black-box attacks on deep neural networks using efficient query mechanisms. In: European Conference on Computer Vision, pp. 158–174. Springer, Berlin (2018)
10. Wu, L., Zhu, Z., Tai, C., et al.: Understanding and enhancing the transferability of adversarial examples (2018). [arXiv:1802.09707](https://arxiv.org/abs/1802.09707)
11. Liu, Y., Chen, X., Liu, C., Song, D.: Delving into transferable adversarial examples and black-box attacks (2016). [arXiv:1611.02770](https://arxiv.org/abs/1611.02770)
12. Papernot, N., McDaniel, P., Goodfellow, I.: Transferability in machine learning: from phenomena to black-box attacks using adversarial samples (2016). [arXiv:1605.07277](https://arxiv.org/abs/1605.07277)
13. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)
14. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582 (2016)
15. Dziugaite, G.K., Ghahramani, Z., Roy, D.M.: A study of the effect of jpg compression on adversarial images (2016). [arXiv:1608.00853](https://arxiv.org/abs/1608.00853)
16. Wu, S., Sang, J., Xu, K., Zhang, J., Sun, Y., Jing, L., Yu, J.: Attention, please! adversarial defense via attention rectification and preservation (2018). [arXiv:1811.09831](https://arxiv.org/abs/1811.09831)
17. Lee, H., Han, S., Lee, J.: Generative adversarial trainer: defense to adversarial perturbations with gan (2017). [arXiv:1705.03387](https://arxiv.org/abs/1705.03387)
18. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: neural image caption generation with visual attention (2015). [arXiv:1502.03044](https://arxiv.org/abs/1502.03044)
19. Yao, L., Torabi, A., Cho, K., Ballas, N., Pal, C., Larochelle, H., Courville, A.: Describing videos by exploiting temporal structure. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 4507–4515 (2015)
20. Chen, Z., Sun, W.: Scanpath prediction for visual attention using ior-roi lstm. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence, pp. 642–648. AAAI Press (2018)
21. Yang, Z., He, X., Gao, J., Deng, L., Smola, A.: Stacked attention networks for image question answering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 21–29 (2016)
22. Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Liu, W., Chua, T.S.: Sca-cnn: Spatial and channel-wise attention in convolutional networks for image captioning. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5659–5667 (2017)
23. Samek, W., Wiegand, T., Müller, K.R.: Explainable artificial intelligence: understanding, visualizing and interpreting deep learning models (2017). [arXiv:1708.08296](https://arxiv.org/abs/1708.08296)
24. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
25. Xie, S., Girshick, R., Dollár, P., Tu, Z., He, K.: Aggregated residual transformations for deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1492–1500 (2017)
26. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
27. Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A.A.: Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-First AAAI Conference on Artificial Intelligence, pp. 630–645 (2017)
28. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: European Conference on Computer Vision, pp. 630–645. Springer, Berlin (2016)
29. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9185–9193 (2018)

30. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)
31. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks (2013). [arXiv:1312.6199](https://arxiv.org/abs/1312.6199)
32. Tai, S.C., Yang, S.M.: A fast method for image noise estimation using laplacian operator and adaptive edge detection. In: 2008 3rd International Symposium on Communications, Control and Signal Processing, pp. 1077–1081. IEEE (2008)
33. Canny, J.F.: A theory of edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **8**, 147–163 (1986)

Chapter 11

Neuron-Level Inverse Perturbation Against Adversarial Attacks



11.1 Introduction

Thus far, intensive research has been devoted to improving DNNs' robustness, along two main directions: reactive and proactive defenses [1]. The former ones apply transformations to the input during the inference stage, while the latter ones modify model structure or training procedure to mitigate attacks. Early proposed reactive strategies, including spatial smoothing [2], JPG compression [3] and total variance minimization [4], attempt to remove perturbations by simple image transformations. These methods [2, 5] cannot handle larger perturbations (e.g., l_2 -norm of perturbation is larger than 0.6 on MNIST). Besides, they require massive adversarial examples for proper parameter training. Meanwhile, other reactive defenses are put forward from the perspective of feature distribution [6, 7], which push distribution of adversarial examples to approximate benign ones via mapping and projection in latent feature space. Their performance is limited by the precision of mapping function and the perturbation size.

Furthermore, proactive defense provides better model robustness. One of the most common approaches is adversarial training [1, 8], which adds adversarial examples of specific attacks to training dataset. It requires a large amount of adversarial examples and high time cost for model retraining. Other proactive methods that develop model structure operations or optimization objectives [9, 10], have been verified their effectiveness but full knowledge of model structures should be given in advance.

Since DNNs' decision results are determined by the non-linear combination of each neuron, some proactive defenses [11] have proved that the misclassification of adversarial examples can be attributed to abnormally activated neurons. Channel-wise activation suppressing (CAS) [12] and sensitive neuron stabilizing (SNS) [13] have connected neuron activation with model robustness. They are both adversarial training strategies based on projected gradient descent (PGD) attack [14], which require massive adversarial examples with high computation cost. Specifically, CAS suppresses redundant neurons according to their contribution during training. But it suffers from the drop of benign accuracy because some class-related neurons are sup-

pressed as well. SNS reduces neuron sensitivity, defined by the changing intensity of neuron activation. But how are these neurons directly related to model predictions hasn't been confirmed from the definition, which finally leads to unsatisfactory defense results.

Based on the above analysis, existing defenses show limitations from three aspects: (1) rely on adversarial examples from specific attacks; (2) can hardly adaptively adjust the defense strategy towards different perturbation sizes; and (3) require high time cost during retraining.

To overcome the above challenges, we reconsider the defense mechanism from the consistency of different attacks and pay special attention to changes in neuron behaviors before and after attacks. Assuming that neurons in the model play different roles in decision-making for classification, we introduce *neuron influence*, which quantitatively measures the extent of neuron's contribution to correct predictions. It is calculated under the guidance of correct labels when the model is fed with benign examples. According to it, neurons in a certain chosen layer are sorted by their influence values in descending order and divided into front neurons (the most class-relevant), tail neurons (the least class-relevant) and the remaining neurons. To further observe neuron behaviors after attacks, corresponding adversarial examples are fed into the model to calculate neuron influence. It can be confirmed that the change of neuron influence follows a regular pattern, i.e., the influence values of front neurons are suppressed while that of tail neurons are enhanced after attacks, regardless of attack methods. Consequently, an important insight is gained that front and tail neurons are easily exploited by different adversarial attacks, which finally causes incorrect predictions of adversarial examples.

Through the neuron activation pattern of general attacks' interpretation, intuitively, it is effective to inverse those maliciously activated and inactivated neurons, to mitigate or even remove the influence of adversarial examples on the model. Inspired by it, we propose an attack-agnostic defense method, dubbed as *Neuron-level Inverse Perturbation*. First, the neuron influence in the chosen layer is calculated by a few batches of benign examples, and then front and tail neurons that may be utilized by adversaries are identified. Next, front neurons are strengthened and tail ones are suppressed, to generate inverse perturbations that can alleviate and offset general adversarial perturbations. Built without any explicit prior knowledge of attacks, our method achieves attack-agnostic defense and can serve as a generic tool. Moreover, by adaptively varying the value of inverse perturbations that match the input example, adversarial examples with different perturbation sizes can be gracefully handled. Besides, without retraining the model, our method only relies on a few benign examples, making it more efficient to deploy defense.

The main contributions are summarized as follows.

- We propose the concept of neuron influence, which is a measurement of the correlation between neurons and ground truth. Then a novel insight is gained that adversarial attacks fool the model by enhancing the least class-relevant neurons with small influence and suppressing neurons with large influence.

- Based on the observation, we design an attack-agnostic defense method. Using benign examples to identify the most and the least class-relevant neurons, enhancing the former, and suppressing the latter ones, our method can mitigate or even eliminate the effect of general adversarial perturbations. Besides, our method can handle adversarial examples with different perturbation sizes adaptively, according to the activation similarity of input examples.
- Extensive experiments on various datasets and models have been conducted. The results show that our method outperforms ($\sim \times 1.3$) the state-of-the-art (SOTA) defense strategies against eleven attacks on image datasets, with only 1/13 computation time on average.

11.2 Related Work

In this section, we review the related work and briefly summarize defense methods used in the experiment. Besides, existing neuron-level approaches for DNNs in security domains are introduced as well.

11.2.1 *Defenses Against Attacks*

Defense techniques could be mainly categorized into two types: reactive defense and proactive defense.

11.2.1.1 **Reactive Defenses**

Reactive methods perform pre-processing operations and transformations on the input images before inputting them to the target model [1]. Dziugaite et al. [3] found that JPG compression could reverse the drop in classification accuracy of adversarial examples to a large extent. Xu et al. [2] reduced the search space available to an adversary by reduction of color bit depth and spatial smoothing. Guo et al. [4] conducted input transformations such as total variance minimization to adversarial examples before feeding them into the model. Based on randomness, these methods are effective but hard to handle larger perturbations. Besides, Mustafa et al. [7] put forward super-resolution as a defense mechanism, which projects adversarial examples back to the natural image manifold learned by models. Similarly, Sun et al. [6] used sparse transformation layer (STL) to achieve the same goal while removing perturbations as well. Liu et al. [5] proposed a DNN-favorable method called feature distillation based on JPG compression, to rectify adversarial examples without the drop of benign accuracy.

11.2.1.2 Proactive Defenses

Proactive defenses involve modifying training data or network structure in the training process. Goodfellow et al. [8] injected adversarial examples into the training set and proposed adversarial training, enhancing the robustness of models. Dhillon et al. [9] developed a stochastic activation pruning method, which prunes the random subset of the activation function and enlarges the remaining ones to compensate. Papernot et al. [15] introduced defense distillation, which uses the knowledge of network to shape its own robustness. As a result, adversarial examples of small perturbations could be resisted. Suri et al. [16] strengthened the robustness of DNNs by targeting neurons that are easily changed by attacks. Bai et al. [12] put forward CAS, which suppresses redundant activation from being activated by adversarial perturbations during training. Zhang et al. [13] explained the adversarial robustness from the perspective of neuron sensitivity, measured by changes in neuron behaviors against benign and adversarial examples. Based on it, they further proposed SNS for retraining the model, finally improving model robustness. Xie et al. [10] added novel blocks that denoise features into convolutional networks for robustness improvement.

11.2.2 *Neuron-Level Approaches for DNN's Security*

Several methods related to neuron behaviors have been put forward for adversarial detection and testing of DNNs.

11.2.2.1 Adversarial Detection

After observing the changes in neuron activation values under various adversarial attacks, Ma et al. [17] proposed the concept of value invariants and the provenance invariants. They extracted them from benign training data and then used them for adversarial detection. Shan et al. [18] injected trapdoors and honeypots into the data, which the attacker's optimization algorithm will fall into. By comparing the signal of input neuron activation and the trapdoor, adversarial attacks can be detected.

These methods select neurons and focus on the changes in activation values before and after attacks for detecting adversarial examples. But they fail to provide fine-grained guidance of rectifying labels after operations, thus they cannot be applied to defense adversarial attacks.

11.2.2.2 Testing Methods for DNNs

Pei et al. [19] proposed DeepXplore, which introduced neuron coverage to systematically measure the extent of DNNs exercised by testing examples. Ma et al. [20] put forward a set of multi-granularity testing standards at a more fine-grained level.

Xie et al. [21] proposed DeepHunter and realized further detection of potential corner cases. It generates mutation examples under the guidance of multiple coverages using mutation-based optimization.

From the view of neurons inside, these works mentioned above achieve exploratory degree of testing for DNNs. Based on neuron coverage, these methods select neurons that can expose more incorrect behaviors, rather than model robustness.

11.3 Preliminaries

11.3.1 Threat Model

A DNN-based classifier can be denoted as $f(x) : X \rightarrow Y$, where $x \in X \subset \mathbb{R}^N$ represents the input and Y denotes predicted classes. We use y to represent ground truth label of x and the adversarial example of x is denoted as x_{adv} .

In order to defend against both white-box and black-box attacks, we define two scenarios of attackers' knowledge. For white-box attacks, attackers have detailed information about model structures and parameters. Besides, they can design adaptive attacks against specific defense approaches. For black-box ones, attackers only know model confidence or predictions, or rely on queries to conduct attacks.

The objective of our method is to defend against unknown attacks without knowing the specific types beforehand, regardless of white-box or black-box attacks. For defenders, they can only access part of correctly classified benign examples with their corresponding labels from training dataset.

11.3.2 Definitions

Given a DNN and let $X = \{x_1, x_2, \dots\}$ denotes a set of input. $N = \{n_1, n_2, \dots\}$ is a set of neurons in the l -th layer. Function $\varphi_n(x)$ represents the output value of neuron $n \in N$, when fed with input $x \in X$.

Definition 11.1 (*Neuron influence*) Neuron influence measures the extent of neuron's contribution to predicted labels. For a neuron $n \in N$, the neuron influence σ of n is defined as

$$\sigma_n = \frac{\partial y^c}{\partial \varphi_n(x)} \cdot \varphi_n(x) \quad (11.1)$$

where y^c denotes the confidence of predicted label c when fed with input x . ∂ denotes partial derivative function. Specifically, neurons with larger σ contribute more to the predicted label c .

Definition 11.2 (*Front neuron, tail neuron and remaining neuron*) According to neuron influence, neurons in the l -th layer are sorted in descending order. *top- k* and *bottom- k* neurons in this neuron sequence are defined as front neurons and tail neurons, denoted as Ω_f and Ω_t , respectively. Other neurons in this layer are called remaining neurons, denoted as Ω_r .

11.4 Methodology

11.4.1 Framework

The framework of our method is shown in Fig. 11.1, including four components: ① neuron selection module, ② generation of neuron template, ③ construction of our method candidates, and ④ adaptivity and reclassification.

Given a test example, neuron selection module is to select front neurons, remaining neurons and tail neurons activated by it in the chosen layer. Neuron template is generated by the remaining neurons from a batch of benign examples. The similarity δ is calculated between neuron template and remaining neurons. The inverse perturbations, generated by front and tail neurons, together with its δ , compose our method's candidates. Inverse perturbation is found via the closest δ from our method's candidates, adjusted adaptively, and then added to the test example, which will be fed back to the model for reclassification. In the following, we will explain the details of each individual component.

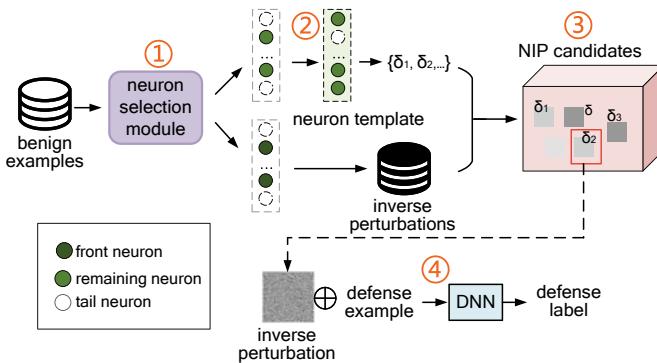


Fig. 11.1 The framework of our method, including four parts: neuron selection module, generation of neuron template, construction of our method's candidates, and reclassification

11.4.2 Neuron Selection Module

Neuron selection module is adopted to select neurons according to their neuron influence. First, the input example is fed into the targeted model. According to Eq. 11.1, neuron influence of each neuron in the chosen layer is calculated. Specifically, the layer between convolutional and dense layers is chosen, e.g., flatten layer or global average pooling. Since this layer contains both pixel features and high-dimensional features, which are crucial for classification. Based on neuron influence, neurons in the chosen layer are sorted in descending order. Front, tail, and remaining neurons, are then divided and selected according to Definition 2. After that, the output neuron sequence is called sorted neurons. They will be further used for generating neuron templates and inverse perturbations.

11.4.3 Generation of Neuron Template

For each model, we generate one neuron template, calculated by a batch of benign examples $X = \{x_1, x_2, \dots, x_t\}$, where t is the number of benign examples. We compare the index of the remaining neurons activated by X , and calculate neuron template T as follows:

$$T = \bigcap_{i=1}^t \{I(\Omega_{r,x_i})\} \quad (11.2)$$

where $I(\Omega_{r,x_i})$ denotes the index of remaining neurons activated by the benign example x_i , and \bigcap denotes the function of taking intersection.

For facilitating the following calculation, we concatenate T with zero vectors, to keep the same dimension as the vector of Ω_r . After that, neuron template is marked as T' .

11.4.4 Construction of Candidates

Our method's candidates consist of inverse perturbation ρ with its corresponding similarity δ . Specifically, δ is calculated by comparing the difference of remaining neurons between inputs and neuron template, while ρ is gained by enhancing front neurons and suppressing tail neurons. Noted in this process, only benign examples are used.

Remaining neurons are used for calculating similarity δ . Given the benign example $x \in X$ and template T' , δ of it can be calculated as follows:

$$\delta_x = \frac{I(\Omega_{r,x}) \cdot T'}{\|I(\Omega_{r,x})\| \times \|T'\|} \quad (11.3)$$

where $\|\cdot\|$ represents l_2 -norm of the vector.

On the other hand, front and tail neurons are used to generate inverse perturbation, which are the *top*- k_1 and *bottom*- k_2 neurons in the sequence of sorted neurons, respectively. k_1 and k_2 are hyper-parameters for selection. The former is responsible for front neurons, while the latter is for tail ones. Inverse perturbation of our method is calculated by enhancing front neurons and suppressing tail ones. We define a loss function to control these neurons activated by x , as follows:

$$\text{loss}_x = \sum \varphi_{n_i}(x) - \sum \varphi_{n_j}(x) \quad (11.4)$$

where $n_i \in \Omega_f$ and $n_j \in \Omega_t$. $\varphi_{n_i}(x)$ denotes the output of the i -th neuron in the front neuron $\Omega_{f,x}$ when fed with input x . Similarly, $\varphi_{n_j}(x)$ denotes the output of the j -th neuron in $\Omega_{t,x}$. \sum denotes the sum function. Opposite to adversarial attacks, loss function strengthens front neurons and weakens tail ones.

Then inverse perturbations are generated by taking the partial derivative of the input, which is calculated as follows:

$$\rho_x = \frac{\partial \text{loss}_x}{\partial x} \quad (11.5)$$

where ρ_x is the inverse perturbation of the input x .

Inverse perturbation is attached to its corresponding δ to compose our method's candidates, as follows:

$$\text{our method's candidates} = \bigcup_{i=1}^t \{\rho_{x_i}, \delta_{x_i}\} \quad (11.6)$$

where δ_{x_i} and ρ_{x_i} denote δ and the inverse perturbation of x_i , respectively. For each benign example $x \in X$ used for construction, we generate and calculate a δ attached to it. So, the number of inverse perturbations with δ is the same as that of benign examples used.

Based on general observations, the generated inverse perturbations are effective against generic attacks. By maintaining candidates of each model, multiple attack-agnostic perturbations can be successfully covered in advance.

11.4.5 Adaptivity and Reclassification

Given a test example \hat{x} , a suitable inverse perturbation ρ_x can be found from candidates according to the closest value of δ_x .

$$\rho_{\hat{x}} \Leftarrow \rho_x \text{ s.t. } |\delta_{\hat{x}} - \delta_x| < \epsilon \quad (11.7)$$

where $\rho_{\hat{x}}$ denotes the suitable inverse perturbation of \hat{x} found by its similarity $\delta_{\hat{x}}$. ρ_x is the inverse perturbation in candidates. ϵ denotes a very small number, i.e., less than 10^{-3} in most cases.

The example after defense can be calculated as follows:

$$x_{def} = \hat{x} + Mean(\hat{x}) \cdot \frac{\rho_{\hat{x}}}{(Max(\rho_{\hat{x}}) - Min(\rho_{\hat{x}}))} \quad (11.8)$$

where x_{def} denotes the example after defense, which will be fed into the targeted model for reclassification. $Mean(\cdot)$, $Max(\cdot)$, and $Min(\cdot)$ denote the average, maximum, and minimum values of the matrix, respectively.

Our method is normalized to match the pixel values of test example in the second half of the formula. Concretely, the pixel value of inverse perturbation will be amplified when the test example contains large perturbations. As a result, the effect of adversarial perturbations of any size can be alleviated by adaptivity of our method.

Theoretically, for each dataset, the number of candidates should be at least the same as the number of categories. More suitably, our method can be found if more candidates are available, which is helpful to better defense performance.

11.4.6 Algorithm Complexity

The complexity of NIP can be divided into two stages: preparation stage and test stage.

In the preparation stage, we select neurons, generate neuron template, and then construct candidates from benign examples. So the computation complexity can be calculated as

$$T_{prepare} \sim \mathcal{O}(t \times a) + \mathcal{O}(t \times a) + \mathcal{O}(t \times a) \sim \mathcal{O}(t \times a) \quad (11.9)$$

where t denotes the number of benign examples and a is the number of neurons in the chosen layer.

In the test stage, we select neurons and calculate similarity to find the corresponding perturbation. Therefore, the time complexity is

$$T_{test} \sim \mathcal{O}(t' \times a) + \mathcal{O}(a) \sim \mathcal{O}(t' \times a) \quad (11.10)$$

where t' denotes the number of test examples.

11.5 Experimental Settings

11.5.1 Datasets

We evaluate our method on three datasets related to image classifications. Experiments are conducted on CIFAR-10, GTSRB and ImageNet on local models.

11.5.2 Models

For CIFAR-10 dataset, VGG19 [22] and AlexNet [23] models are adopted. LeNet-5 [24] and ResNet20 [25] are used for the GTSRB dataset. For ImageNet, VGG19 and MobileNetV1 [26] are used.

11.5.3 Attacks

We use eleven attack methods to generate adversarial examples of different perturbation sizes and distributions, which are commonly used to test defense performance. White-box attacks include FGSM [8], BIM [27], MI-FGSM [28], JSMA [29], PGD [14], DeepFool [30], and UAP [31]. Black-box attacks contain AUNA [32], PWA [33], One pixel (Pixel) [34], and Boundary [35]. Attack success rate and average perturbation sizes are shown in Fig. 11.2, where ϵ denotes the average perturbation size after normalization to $[0, 1]$. Pixel attack is not applied for ImageNet and VCTK because it does indeed lower the confidence but still fails to fool the model for all conducted examples. AUNA attack with high transferability is adopted to craft adversarial examples for online platforms, with ASR 100% for both self-trained substitute models.

11.5.4 Defense Baselines

For image classification, ten defense methods are adopted in our experiment. Seven reactive defenses are conducted, including ① spatial smoothing (SS) [2], ② color bit-depth reduction (CBDR) [2], ③ JPG compression (JC) [3], ④ total variance minimization (TVM) [4], ⑤ STL [6], ⑥ super resolution (SR) [7], and ⑦ feature distillation (FDistill) [5]. Besides, ⑧ CAS [12], ⑨ SNS [13], and ⑩ feature denoising (FDenoise) [10], categorized as proactive defenses, are also adopted as baselines.

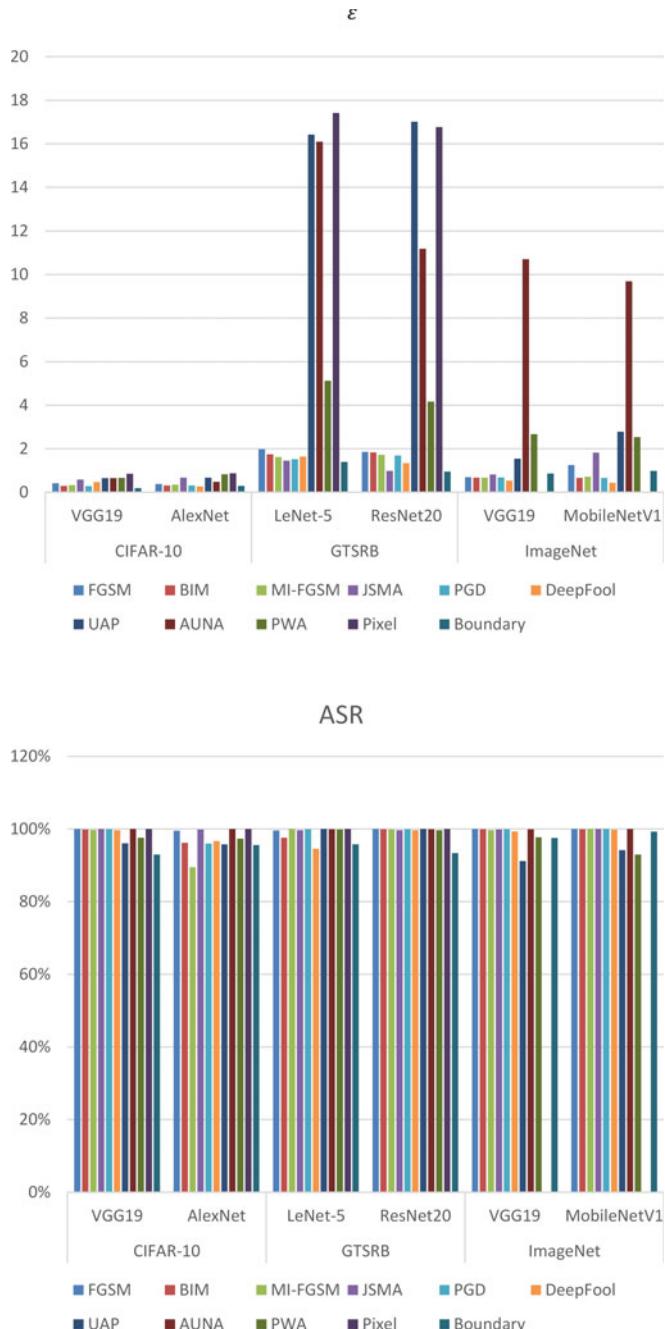


Fig. 11.2 The details of adversarial examples. The pixel values of each example are normalized to [0,1]. ϵ denotes the l_2 norm of perturbations and ASR denotes the attack success rate

11.5.5 Evaluation Metrics

The metrics used in the experiments can be divided into two aspects, one for evaluating defense performance, the other for interpretation of defense. The former includes classification accuracy (acc), perturbation l_2 -norm, attack success rate (ASR), and defense success rate (DSR). The latter contains $top-k$ neuron consistency (TKNC) and peak signal to noise ratio (PSNR) [36].

① Classification accuracy: $acc = \frac{N_{true}}{N_{benign}}$, where N_{true} is the number of benign examples correctly classified by the targeted model and N_{benign} denotes the total number of benign examples.

② Perturbation l_2 -norm: $\epsilon = \|x_{adv} - x\|_2$, where x and x_{adv} are benign and its corresponding adversarial example, respectively, and $\|\cdot\|_2$ represents l_2 -norm. l_2 -norm is a common measure to evaluate the average size of the perturbations in the adversarial examples.

③ Attack success rate: $ASR = \frac{N_{adv}}{N_{benign}}$, where N_{adv} denotes the number of adversarial examples.

④ Defense success rate: $DSR = \frac{N_{def}}{N_{adv}}$, where N_{def} denotes the number of adversarial examples correctly classified by the targeted model after defense.

⑤ $top-k$ neuron consistency: $TKNC(X, X', i, k) = \frac{\text{top}_k(X) \cap \text{top}_k(X')}{\text{top}_k(X) \cup \text{top}_k(X')}$, where X denotes a batch of benign examples and X' denotes that of adversarial examples or examples after defense. top function selects $top-k$ important neurons in the i -th layer, activated by X and X' . TKNC measures the consistency of important neurons. A greater value denotes higher consistency.

⑥ Peak signal to noise ratio: $PSNR = 20 \times \log_{10} \frac{\max_I^2}{MSE}$. MSE can be calculated as $MSE = \frac{1}{m \times n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I - K]^2$, where I and K are two images with the same size $m \times n$. \max_I^2 is the possible maximum pixel value of image I . Images with larger PSNR have better image quality and smaller distortions.

11.6 Evaluation and Analysis

In this section, we evaluate the performance of our method by answering the following several research questions (RQs):

- **RQ1:** Does our method perform better defense effect against various attacks, when compared with SOTA baselines?
- **RQ2:** Can our method show stable defense via adaptivity against different perturbation sizes?
- **RQ3:** From the perspective of neuron inside, how efficient is our method in defending attacks?

11.6.1 RQ1: Effectiveness

When reporting the results, we focus on the following aspects: defense against various attacks and its impact on benign examples.

11.6.1.1 Defense Results Against Various Attacks

We focus on the defense results of our method to measure the effectiveness against various adversarial examples.

Implementation Details. (1) Six models of image classification are adopted and 5,000 adversarial examples per attack are used for evaluation. ϵ and ASR of these adversarial examples are shown in Fig. 11.2. (2) For measurement, DSR is calculated on all models after defense. Besides, the p-value of t-test between our method and each baseline is also calculated. Two-tailed and paired-sample t-test is adopted for calculation. (3) Fig. 11.3 shows DSR of our method and baselines against different attacks, where defense performance of our method is denoted by brown lines, while other methods are represented by scattered points with different shapes and colors. And their p-values are reported in Fig. 11.4, where p-value larger than 0.05 is bold in the table.

Results and Analysis. In all cases, our method significantly improves the model's classification towards adversarial attacks, with its DSR almost up to 100%, especially on CIFAR-10 and ImageNet. Besides, it outperforms both proactive and reactive defense baselines with a considerable margin. For instance, in Fig. 11.3, almost all the DSRs are the highest. And, almost all p-values of our method in Fig. 11.4 are less than 0.05. The outstanding performance of our method is mainly because it can alleviate and even offset the effect of adversarial perturbations, by strengthening front neurons and weakening tail ones that are exploited by attacks.

More specifically, when dealing with larger perturbations such as AUNA and UAP, our method shows a slightly inferior defense effect to that of smaller ones. For instance, DSR of AUNA on MobileNetV1 of ImageNet is around 90%, inferior to others about 99%. One possible reason is that large perturbations to some extent destroy some important features related to correct classification, leading to the decrease in DSR.

Our method shows more stable defense performance among various adversarial attacks, regardless of attack types or image scales. On the contrary, fluctuations can be observed among baselines over different attacks, especially black-box ones. For instance, on AlexNet of CIFAR-10, DSR of our method is around 98% on average against various attacks, while DSR of SNS shows sharp decrease on PWA. We speculate the possible reason is that our method implements defense based on general observations of neuron behaviors, so it can deal with perturbations from various attacks smoothly.

With regard to multi-class dataset like GTSRB, the effectiveness of our method is also guaranteed on a majority of attacks. This is because our method links adversarial

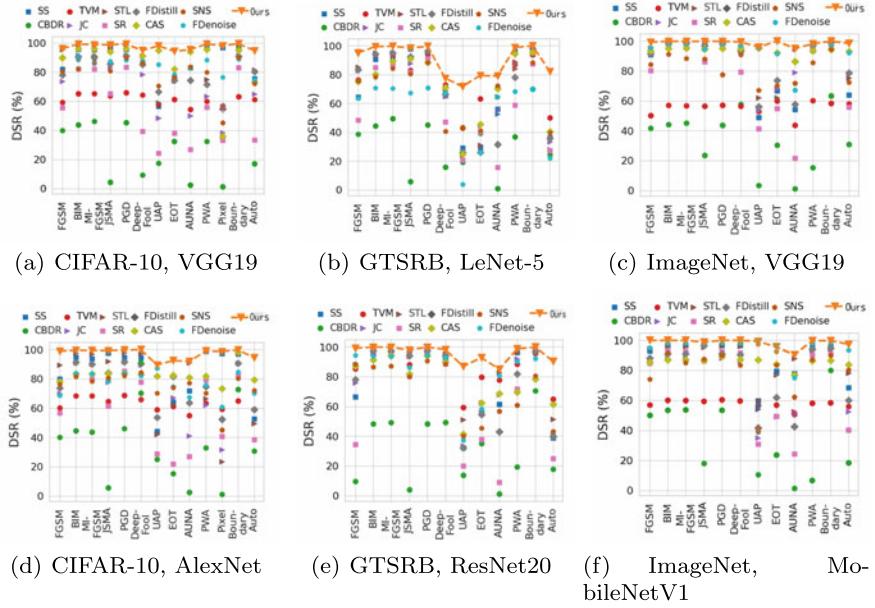


Fig. 11.3 Comparison of defense results against white-box and black-box attacks on different datasets and models

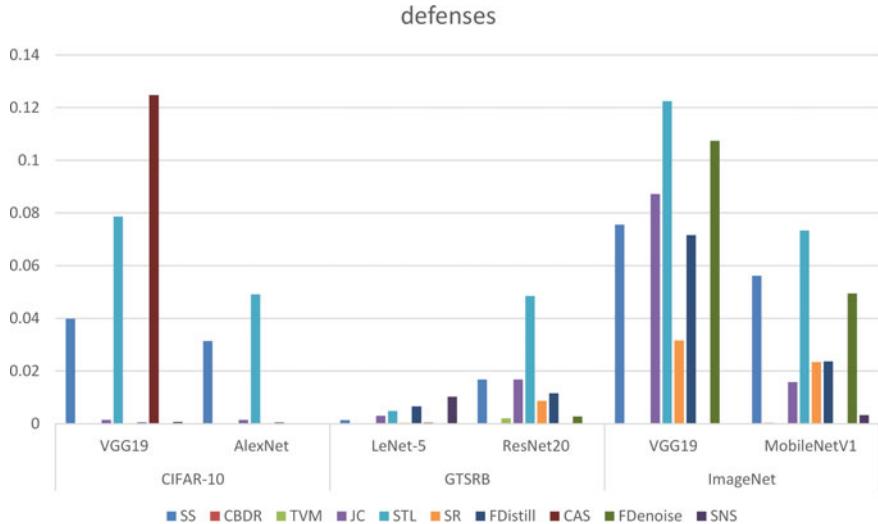


Fig. 11.4 The p-value of t-test between our method and each baseline. Values larger than 0.05 are bold

examples with corresponding inverse perturbation. That is more fine-grained solution since we nearly generate our method tailored to each input image. It must be noted that on GTSRB, DSR of UAP and Pixel show a sharp decrease. We investigate the reason and find that perturbations added are so large that a human can hardly tell the contents in these examples. Important details are mostly destroyed, making it harder for providing correct labels.

11.6.1.2 Defense Impact on Benign Examples

Since some of the existing defenses are compromised to the accuracy of benign examples, in this part we focus on the impact of our method on benign examples.

Implementation Details. (1) Six models, same as Sect. 11.6.1.1, are used for evaluation. (2) 2,000 benign examples are randomly selected from each dataset and benign accuracy after our method and ten baselines will be measured. (3) Results are shown in Fig. 11.5.

Results and Analysis. From Fig. 11.5, we can conclude that our method does not sacrifice the classification accuracy of benign examples while completing efficient defense. In Fig. 11.5, benign acc of our method in all cases is up to 99% on average. This is because front neurons related to correct classifications are strengthened, which helps our method maintain high accuracy on benign examples. With regard to baselines, benign accuracy of CAS shows inferior result (around 94%), for it suppresses some class-relevant neurons repeatedly during training.

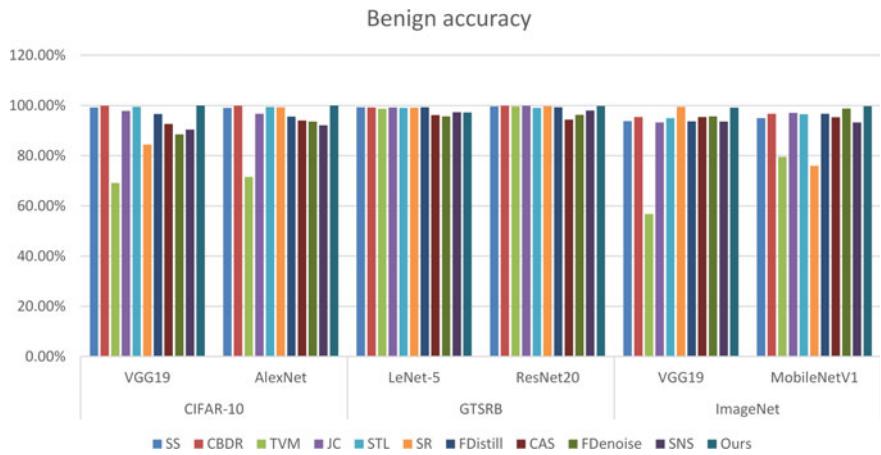


Fig. 11.5 Benign accuracy of our method and defense baselines

Answer to RQ1: Our method outperforms the SOTA baselines in two aspects: (1) DSR against various attacks ($\sim \times 1.3$ on average); (2) benign accuracy—it maintains the highest benign accuracy (i.e., almost up to 99% on average).

11.6.2 RQ2: Generality

When reporting the results, we focus on defense generality on different perturbation sizes.

Implementation Details. (1) VGG19 of CIFAR-10 and MobileNetV1 of ImageNet are adopted. One thousand adversarial examples generated by PGD are used. l_2 -norm perturbation size ϵ from 0.2 to 1.6 added on CIFAR-10, while ϵ from 0.4 to 2.2 for ImageNet. (2) SS, FDistill, and STL are chosen as baselines due to their superior defense effect, according to Sect. 11.6.1.1. (3) The experimental results per ϵ are shown in Fig. 11.6, where broken lines of navy, light blue, green, and orange denote DSR of SS, FDistill, STL, and our method, respectively.

Results and Analysis. Our method shows stronger robustness on defense effect than baselines, among all perturbations on both datasets. For instance, orange lines are the highest in Fig. 11.6. With the increase in perturbation size, a slight drop in DSR of our method can be observed. Specifically, when ϵ reaches 1.6, baselines (i.e., STL and SS) can hardly take effect on CIFAR-10, with DSR around 30%. But DSR of our method

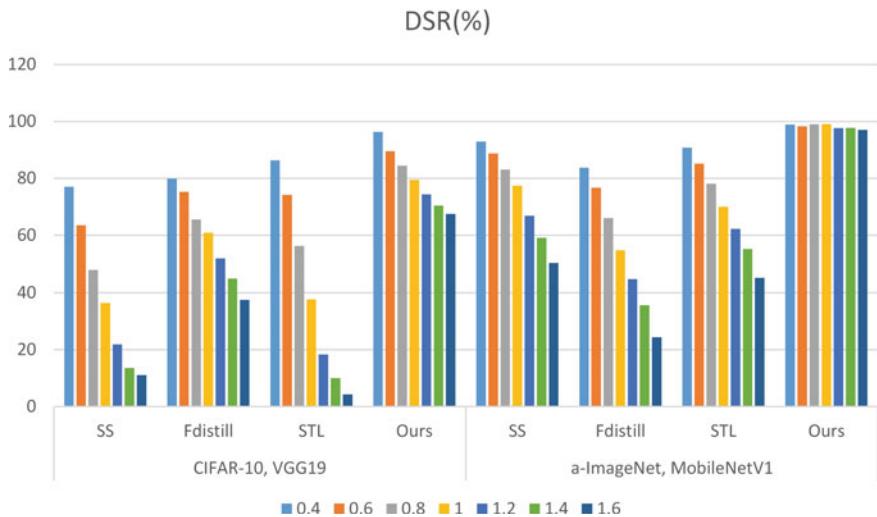


Fig. 11.6 DSR of our method and three baselines under different perturbation sizes, where our method shows more stable performance than baselines

still exceeds 70%. It is mainly because adversarial perturbations of different sizes can be alleviated and removed by our method via dynamically and adaptively adding inverse perturbations. This well demonstrates the effectiveness of adaptivity before reclassification. However, defense based on feature distribution are compromised in the event of large perturbations, let alone simple image transformations. They fail to restore the feature distribution of benign examples, leading to unsatisfactory results.

Answer to RQ2: Compared with baselines, our method shows quite stable DSR ($\sim \times 3.4$ in the worst case) via adaptivity against adversarial perturbations of different sizes. This verifies its generality on perturbation sizes.

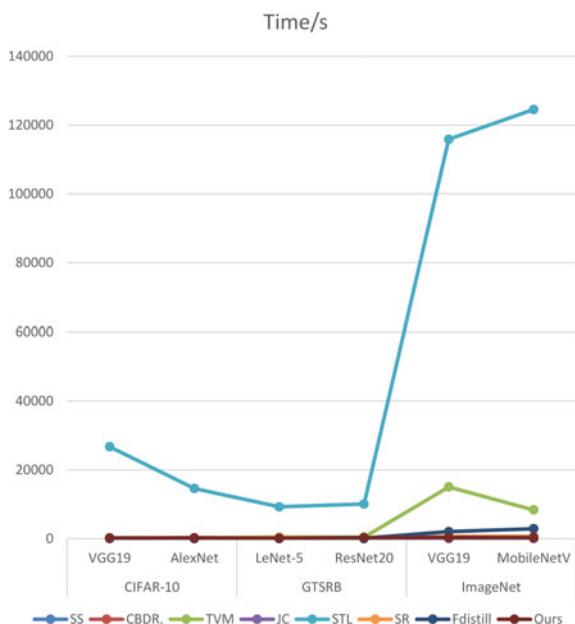
11.6.3 RQ3: Efficiency

In this part, we give a comparison of defense time between our method and baselines.

Implementation Details. (1) The average defense time of our method and reactive baselines are calculated in the test stage. We measure the average running time on 5,000 adversarial examples crafted by each attack mentioned in Sect. 11.5 for 3 times, and the minimal one is identified as the final result. (2) Fig. 11.7 shows the results.

Results and Analysis. From Fig. 11.7, running time of our method is acceptable, especially on large dataset like ImageNet. For instance, time of our method on

Fig. 11.7 Comparison of time complexity between our method and reactive baselines in the testing stage



CIFAR-10 is around 200 s, inferior to that of SR and FDistill slightly. But on ImageNet, our method requires much less time (around 300 s) than TVM (around 1000 s), STL (over 10000 s), and FDistill (over 2000s). In the testing stage, our method searches for the closest inverse perturbation. So, the computation time of it is only related to the number of examples under defense. For baselines like STL and FDistill, more computation time is required on large images for they contain more pixels. But compared with simple image transformation like SS, CBDR, and JC, our method shows disadvantage. This is mainly because it involves the model structure and traverses all neurons in the selected layer to calculate neuron influence. The algorithm complexity of our method has been analyzed in Sect. 11.4.6.

Answer to RQ3: In the test stage, our method achieves defense using 1/13 computation time on average, compared with reactive baselines. Although inferior to simple image transformations on time complexity, it still works efficiently with acceptable time cost, especially on large datasets.

11.7 Conclusion

In this chapter, we introduce the concept of neuron influence and then divide neurons into front, tail, and remaining neurons. We observe that multiple attacks fool the model by suppressing front neurons and enhancing tail ones. Motivated by it, we propose a new attack-agnostic defense method, which in turn strengthening front neurons and weakening the tail part. As a result, it outperforms the state-of-the-art defense baselines on various datasets and models.

References

1. Mustafa, A., Khan, S.H., Hayat, M., Goecke, R., Shen, J., Shao, L.: Deeply supervised discriminative learning for adversarial defense. *IEEE Trans. Pattern Anal. Mach. Intell.* 1–13 (2020)
2. Xu, W., Evans, D., Qi, Y.: Feature squeezing: detecting adversarial examples in deep neural networks. In: 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018. pp. 1–15. The Internet Society (2018)
3. Dziugaite, G.K., Ghahramani, Z., Roy, D.M.: A study of the effect of jpg compression on adversarial images. *arXiv preprint arXiv:1608.00853* pp. 1–8 (2016)
4. Guo, C., Rana, M., Cisse, M., van der Maaten, L.: Countering adversarial images using input transformations. In: International Conference on Learning Representations, pp. 1–12 (2018)
5. Liu, Z., Liu, Q., Liu, T., Xu, N., Lin, X., Wang, Y., Wen, W.: Feature distillation: Dnn-oriented jpeg compression against adversarial examples. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 860–868. IEEE (2019)

6. Sun, B., Tsai, N.H., Liu, F., Yu, R., Su, H.: Adversarial defense by stratified convolutional sparse coding. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11447–11456 (2019)
7. Mustafa, A., Khan, S.H., Hayat, M., Shen, J., Shao, L.: Image super-resolution as a defense against adversarial attacks. *IEEE Trans. Image Process.* **29**, 1711–1724 (2019)
8. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings, pp. 1–11 (2015)
9. Dhillon, G.S., Azizzadenesheli, K., Lipton, Z.C., Bernstein, J., Kossaifi, J., Khanna, A., Anandkumar, A.: Stochastic activation pruning for robust adversarial defense. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings. pp. 1–13. OpenReview.net (2018)
10. Xie, C., Wu, Y., Maaten, L.V.D., Yuille, A.L., He, K.: Feature denoising for improving adversarial robustness. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 501–509 (2019)
11. Liu, Y., Lee, W., Tao, G., Ma, S., Aafer, Y., Zhang, X.: ABS: scanning neural networks for backdoors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11–15, 2019, pp. 1265–1282. ACM, New York (2019)
12. Bai, Y., Zeng, Y., Jiang, Y., Xia, S.T., Ma, X., Wang, Y.: Improving adversarial robustness via channel-wise activation suppressing. In: International Conference on Learning Representations, pp. 1–19 (2020)
13. Zhang, C., Liu, A., Liu, X., Xu, Y., Yu, H., Ma, Y., Li, T.: Interpreting and improving adversarial robustness of deep neural networks with neuron sensitivity. *IEEE Trans. Image Process.* **30**, 1291–1304 (2020)
14. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018. pp. 1–28. OpenReview.net (2018)
15. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22–26, 2016. pp. 582–597. IEEE Computer Society (2016)
16. Suri, A., Evans, D.: One neuron to fool them all. arXiv preprint [arXiv:2003.09372](https://arxiv.org/abs/2003.09372) pp. 1–5 (2020)
17. Ma, S., Liu, Y.: Nic: Detecting adversarial samples with neural network invariant checking. In: Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019), pp. 1–15 (2019)
18. Shan, S., Wenger, E., Wang, B., Li, B., Zheng, H., Zhao, B.Y.: Gotta catch'em all: Using honeypots to catch adversarial attacks on neural networks. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 67–83 (2020)
19. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: automated whitebox testing of deep learning systems. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 1–18 (2017)
20. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y., et al.: Deepgauge: multi-granularity testing criteria for deep learning systems. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 120–131 (2018)
21. Xie, X., Ma, L., Juefei-Xu, F., Xue, M., Chen, H., Liu, Y., Zhao, J., Li, B., Yin, J., See, S.: DeepHunter: a coverage-guided fuzz testing framework for deep neural networks. In: Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 146–157 (2019)
22. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–14 (2015)

23. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3–6, 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114 (2012)
24. LeCun, Y., et al.: Lenet-5, convolutional neural networks **20**(5), 14 (2015). <http://yannlecun.com/exdb/lenet>
25. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
26. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint [arXiv:1704.04861](https://arxiv.org/abs/1704.04861) pp. 1–9 (2017)
27. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, pp. 1–14. OpenReview.net (2017)
28. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 9185–9193 (2018)
29. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 372–387. IEEE (2016)
30. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 2574–2582. IEEE Computer Society (2016)
31. Moosavi-Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1765–1773 (2017)
32. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Additive uniform noise attack in foolbox tool
33. Schott, L., Rauber, J., Bethge, M., Brendel, W.: Towards the first adversarially robust neural network model on MNIST. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019, pp. 1–16. OpenReview.net (2019)
34. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5), 828–841 (2019)
35. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: reliable attacks against black-box machine learning models. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–12. OpenReview.net (2018)
36. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**(4), 600–612 (2004)

Chapter 12

Adaptive Channel Transformation-Based Detector for Adversarial Attacks



12.1 Introduction

In the past decade, deep learning has played an increasingly important role in decision-making in the field of computer vision. Computer vision [1], traffic prediction [2], and bioinformatics analysis [3]. The training of Convolutional Neural Networks (CNNs) has become easier due to the improvement of open-source deep learning frameworks and hardware performance. Consequently, more and more deep learning models based on convolutional neural networks and their variants have been proposed in recent years, including AlexNet [4], VGG [5], Fast/Faster R-CNN [6], and ResNet [1]. However, as deeper research has been conducted on deep neural networks (DNNs), it has been found that DNNs are susceptible to attacks by hostile instances [7]. An example of spoofing behavior when DNN-based systems are threatened is face recognition systems. Face Recognition System (FRS), [8], causing self-driving cars to misrecognize road signs [8] or community spoofing [9]. All of them can create serious security problems. In order to detect the security threats in DNNs, many methods have been proposed to counter the attacks. Based on the attacker's knowledge of DNNs, adversarial attacks can be categorized into white-box, black-box, and gray-box attacks [10]. A number of defensive methods [11] have been proposed to remedy the vulnerabilities of DNNs and improve their robustness. The different implementation techniques of defense operations can be categorized as training/input modification [12], network modification [13], and network-attached defense [14]. The defenses can be categorized into re-identification defenses and detection defenses depending on the purpose of the defense effect. The majority of detection methods are based on the difference between adversarial and clean samples. On a theoretical basis, the detecting performance is improved and the complexity of the process is increased when many different features are selected. But in practice, it has been found that different features contribute differently to the detection results, and some features even have a negative impact.

Valid identification of key features is one of the main issues in detection defense. Several data transformation-related detection methods based on data transformation

have been covered in previous work. Nevertheless, there are still some challenges to overcome: (1) The transformation type selection and parameterization are more based on experience, (2) The excessive number of transformation channels increases the computational complexity, and (3) The detection effect of different transformation channels is redundant. Hence, it would be an essential improvement to the detection technique if the appropriate transform types and parameters can be determined adaptively, which would be an important improvement for the application of data transform-based detection methods. Motivated by this fact, we propose a lightweight detector based on adaptive channel transform, namely our method, to answer the following questions: First of all, how to identify the key features that contribute significantly to the detector so as to realize a lightweight detector while maintaining the detection effectiveness? To address this question, we employ the Binary Cuckoo Search (BCS) algorithm to identify near-optimal features. Secondly, how to speed up the search of key features and shorten the training time of the detector? To overcome this challenge, we design a fast training strategy based on the random weight parameter sharing and a Pareto-based multi-objective optimization strategy. Both of these methods are guaranteed to converge to a near-optimal solution. Thirdly, what is the best way to achieve attack detection defense and detection transferability for deep models? Namely, a detector trained on adversarial examples generated by one attack can detect adversarial examples generated by another attack. Moreover, a detector which is trained based on adversarial examples obtained by attacking one of the models can also be able to detect adversarial examples generated by attacking another model. We use an ensemble training strategy, i.e., the detector is trained using adversarial examples from multiple attacks against multiple deep models. As a result, the detector can endure examples from unknown attacks and unknown models. In addition, we also find that the ACT detector can detect the types of attacks against adversarial examples, thus providing guidance on the reidentification defense that should be applied. Our main contributions are summarized below:

1. Our method, an adaptive channel transformation-based detection defense optimization framework, is designed to optimize channel type and number while ensuring detection effectiveness. Our method can identify the critical feature channels that make a significant contribution to the detection result, thus enabling lightweight adversarial examples.
2. In order to accelerate the channel optimization process and ensure the detection efficiency, in this chapter, a Pareto-based multi-objective optimization strategy is proposed to achieve the fast convergence of the global optimal solution. In addition, in this chapter, a fast detector training method based on random sharing of weight parameters during cuckoo search (CS) is proposed, which reduces the computation time of the fitness values.
3. Extensive experiments demonstrate the detection effectiveness of our method. It is shown that the proposed method can detect not only adversarial instances and normal instances but also the attack types of adversarial instances. The five near-optimal channels automatically selected by our method can achieve detection results comparable to the 45C-Detector, which provides defense transferability for different depth models.

12.2 Related Works

In this section, we review the relevant literature and briefly summarize the attack and defense techniques used in the experiments. In addition, we investigate related optimization methods.

12.2.1 Detection Defenses

A broad range of defense techniques have been proposed to deal with adversarial threats. The defense methods can be classified into two categories depending on the purpose of the classification: complete defense and detection-only defense. Former can be further categorized into two approaches [10]: the first is to defend against adversarial attacks by modifying the network structure, while the second is to switch the input examples or change the training during the learning process. The latter avoids tuning the original network structure by using external detectors to classify adversarial examples.

In the initial stages of adversarial research, several approaches [15] mention the use of adversarial training against attacks, which does improve the robustness of DNNs. These models, however, need to be trained using adversarial examples generated for a specific attack type; that is, these techniques provide a passive defense. In spite of this, it is undeniable that adversarial training has inspired the following research. As opposed to modifying the training process or the input data, certain methods alter the network. For instance, Ross and Doshi-Velez proposed to improve adversarial robustness by regularizing the input gradient. A defensive distillation method was introduced by Papernot et al. [13] to extract additional knowledge about the training points as feedback to the training process.

Specifically, the second category involves building a detector that is capable of distinguishing between adversarial examples. A “sensitivity” measure was proposed by Wang et al. The authors integrate statistical model checking and mutation testing by measuring sensitivity to determine whether input examples are likely to be clean or malicious at runtime. It was found that the adversarial examples did not exactly match the saliency. Meng et al. [16] suggested MagNet, a framework for defending neural network classifiers against adversarial examples. MagNet includes one or more independent detector networks and a reformer network. Based on a computationally efficient image enhancement method, Mustafa et al. [17] provided a robust defense technique that effectively mitigates the impact of adversarial attacks.

An external detector to recognize adversarial examples is an effective option to ensure classification accuracy and prevent the model from being modified. Tian et al. [18] demonstrated that adversarial examples are always sensitive, while clean examples are generally immune to transformation operations, and they employed 45 transformation operations to detect adversarial examples. However, transforming each example into 45 channels takes extra time, which adds to the complexity of

the DNN. Furthermore, the transform types and parameters are empirically set and cannot be determined adaptively. Experiments have found that different channels should be used for different datasets to obtain better detection results. For example, the rotation transformation needs to be set at different angles [18] ($[-30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ]$ for MNIST and $[-50^\circ, -25^\circ, 0^\circ, 25^\circ, \text{and } 50^\circ]$ for CIFAR10) to achieve a higher detection rate. In addition, the detection performance remains the same after removing some of the proposed 45 channels. In other words, the complexity can be reduced by optimizing the channels. Therefore, we design a channel optimization strategy that automatically selects the near-optimal transform with guaranteed detection performance, and reduces the time complexity and space complexity of the detection process.

12.3 Methodology

This section describes the details of our method. First, the framework of our method is introduced. Then, candidate channel transforms are prepared for adaptive channel optimization. Subsequently, the CS algorithm is used to select the type and number of near-optimal channel transforms. Finally, the lightweight detector structure and fast training strategy are introduced.

Our method consists of three parts: generation of candidate channel transforms, BCS-based adaptive channel transform (ACT), and lightweight detector training, as shown in Fig. 12.1.

First, candidate channel transform images are generated. The original image is subjected to K kinds of channel transforms to obtain the transformed candidate images $\{x^{CT,1}, x^{CT,2}, \dots, x^{CT,j}, \dots, \text{and } x^{CT,K}\}$, where $x^{CT,0}$ is the original image. Since the solution of BCS is based on binary coding, the quality of candidate images affects the optimization results. The generated candidate images determine the size of the search space, which should be diverse and comprehensive. Therefore, we design as many transformation types and parameters as possible, as detailed in Sect. 12.3.2.

12.3.1 Framework

After that, ACT optimization is achieved on top of BCS, which is the key step in our method. The candidate-transformed image is input into the classifier to generate the confidence matrix ${}^c x \in \mathbf{R}^{(K+1) \times N}$. The confidence matrix is ${}^c x = ((y^{CT,0})^T, (y^{CT,1})^T, \dots, (y^{CT,j})^T, \dots, \text{and } (y^{CT,K})^T)^T$. To obtain the optimization confidence matrix ${}^{oc} x$, each row element of ${}^c x$ is multiplied by each row element of $[net_s(t)]^T$, which is defined as

$${}^{oc} x_i(jr, jc) = {}^c x_i(jr, jc) \times [net_s(t)]^T(jr, 0), \quad (12.1)$$

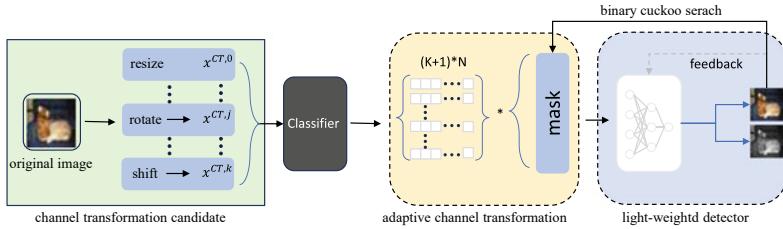


Fig. 12.1 Framework of our method based on BCS. First, the original image is implemented with K types of channel transformations to generate the candidates $\{x^{CT,1}, x^{CT,2}, \dots, x^{CT,j}, \dots, x^{CT,K}\}$, where $x^{CT,0}$ is the original image. Thereafter, the transformed images in the candidates are input into the classifier to obtain the confidence matrix ${}^c x$, with a dimension of $(K+1) * N$. Subsequently, each row element of the confidence matrix ${}^c x$ is multiplied by each row element of the binary solution $[net_s(t)]^T$, thereby selecting approximately optimal channel transformations. Finally, the optimized confidence matrix ${}^{oc} x$ is input into the lightweighted detector to determine whether the original image is an adversarial example. To accelerate the CS, a rapid training strategy is applied to the detector

where ${}^c x_i(jr, jc)$ represents the element of ${}^c x_i$ in column jc and row jr . ${}^{oc} x_i$ has the same dimensions as ${}^c x_i$. Moreover, $[net_s(t)]^T(jr, 0)$ represents the element of $[net_s(t)]^T$ in row jr , $net_s(t) \in \mathbf{R}^{1 \times (K+1)}$ is the s -th solution of the BCS at the t -th iteration, and $[net_s(t)]^T$ is the transpose of $net_s(t)$. The detailed BCS optimization operations are presented in Sect. 12.3.3.

Then, the lightweight detector is trained, including the training dataset, detector structure, loss function, and training hyperparameters. The clean example is denoted as $X = \{x_0, x_1, x_2, \dots, x_i, \dots, x_{m_c-1}\}$, and the confrontation example is expressed as $X^* = \{x_0^*, x_1^*, x_2^*, \dots, x_i^*, \dots, x_{m_{adv}-1}^*\}$, where m_c and m_{adv} are the number of clean examples and adversarial examples, too. The adversarial examples used in the experiment can be realized $\hat{l}^* \neq l$. The confidence matrices corresponding to x_i and x_i^* are denoted as ${}^c x_i \in {}^c X$ and ${}^c x_i^* \in {}^c X^*$, respectively, and they constitute the training dataset for the detector. The fundamental truth values of ${}^c X$ and ${}^c X^*$ are defined as 0 and 1, respectively, as shown in Fig. 12.1. For more details on detector training, see Sect. 12.3.4.

Lastly, an approximate optimal confidence matrix ${}^{oc} x_i$ is computed for each x_i based on the global optimal solution net_{global} . At this stage, ${}^{oc} x_i$ is fed into the detector to determine if the original image x_i is an adversarial example.

12.3.2 Channel Transformation Candidates

The channel transitions investigated in this study include resizing, rotating, shifting, and noise, which can effectively increase the diversity of the channel candidates.

As the adversarial examples and the pure examples show different features after some transformations, hence, they have a high detection success rate. To be more

specific, different transformation methods contribute differently to the detection. The number of total channel transformations in our method is close to that of Tian [18], but with more channel types and parameters, representing a more comprehensive search space.

In selecting the candidate transforms, we mainly consider the contribution of each transform based on the detection results. We train a single-channel detector for each transform (e.g., resize +2). We use it as a candidate if the detection rate is higher than 55%. The reason for this is that a detection rate higher than 55% indicates that the detector can detect adversarial examples based on the differences shown by the transforms rather than random detection.

Too many channel transforms increase the complexity of the detector space and the difficulty of data preprocessing. But this only serves to increase the computational complexity of the training process. Furthermore, we have designed a speedup for the training process to minimize the effect of the increase in the number of channels on the computational complexity. Various transform parameters contribute differently to the detection effect and can even hinder the detection of adversarial examples. Hence, channel transforms should be optimized to determine the type and number of near-optimal ones.

12.3.3 ACT Based on BCS

Various transformations contribute to the detection results in different ways. Some transformations can have even a negative impact on the detection rate. In addition, too many transforms will increase the difficulty of the complexity of the detector parameters and data preprocessing. Hence, we propose a BCS-based ACT with the objective of using as few transform as possible types of transformations while maintaining the detection rate. Two main steps are involved: a BCS-based near-optimal transform solution and a Pareto-based multi-objective optimization.

12.3.3.1 Approximately Optimal Transformation Solutions Based on BCS

- **Initialization of Nests**

The BCS uses a random initialization process, which is defined as follows:

$$\begin{aligned} nest_i^j(0) &= \text{rand}\{0, 1\} \\ i &= \{0, 1, 2, \dots, N_{\text{nest}} - 1\}, j = \{0, 1, 2, \dots, N_{\text{dime}} - 1\}, \end{aligned} \quad (12.2)$$

where $nest_i^j(0)$ represents the value of the j -th dimension of the i -th solution at the 0-th iteration, N_{dime} is the dimension of each solution, $\text{rand}\{0, 1\}$ randomly generates

0 or 1, and N_{nest} is the number of solutions. In the experiment, $N_{nest} = 25$ and $N_{dime} = 55$.

• Multi-objective Fitness Functions

The two objective functions are defined as f_1 and f_2 , which represent the inverse of the detection rate and the number of channel transformations, respectively. The fitness function of each solution $nest_i(t)$ is defined as follows:

$$f_1(nest_i(t)) = \frac{1}{DR}, \quad f_2(nest_i(t)) = \sum_{j=0}^{N_{dime}-1} nest_i^j(t), \quad (12.3)$$

where $DR = (TP + TN)/(TP + TN + FP + FN)$ is the detection rate, $\sum_{j=0}^{N_{dime}-1} nest_i^j$ is the number of 1s in the solution $nest_i(t)$, and t is the number of iterations. Furthermore, TP, TN, FP, and FN represent true positive, true negative, false positive, and false negative, respectively. In our experiments, the confrontation paradigm and the clean paradigm are defined as positive and negative paradigms, respectively.

This is a multi-objective optimal problem. In this problem, we want the detector to have a high detection rate and a low number of channel transformations. Nevertheless, the detection rate will decrease as the number of channel transformations decreases. Hence, we search for the trade-off position between f_1 and f_2 based on the Pareto curve.

• Updating Nest Positions

We employ the following equations to update each nest.

$$nest'^j_i(t) = \begin{cases} 1, & \text{if } S(\phi) > \sigma \\ 0, & \text{otherwise} \end{cases}, \quad (12.4)$$

where $\sigma \sim U(0, 1)$, $U(\cdot, \cdot)$ represents a uniform distribution, returning a value between 0 and 1 and $S(\cdot)$ is the sigmoid function representing the probability of updating the nest. Here, $S(\cdot)$ and ϕ are defined as

$$S(\phi) = \frac{1}{1 + e^{-\phi}}, \quad \text{where } \phi = nest_i^j(t-1) + \alpha \oplus Levy(\lambda), \quad (12.5)$$

where \oplus indicates entry-wise multiplications, α is the step size scaling factor, $Levy(\cdot)$ is the Levy flight, which applies a random walk by Levy distribution, and $1 < \lambda \leq 3$.

After the generation of $nest'(t)$, we get a population of size 2/times N_{nest} , including $nest'(t)$ and $nest(t-1)$. We choose a single new population $nest(t)$ of size N_{nest} by following a non-dominated ordering hierarchy of sorting. After that, this new population will undergo a desertion process so that each nest in the population has the chance to change its endogenous value.

- **Replacing Worst Nests**

To make sure that the new scheme consists of good nestings, we will eliminate the poor nestings based on the probability of p_a . After that, we will replace the worst nesting with the new scheme according to the following equation:

$$nest_i(t) = \begin{cases} 1 - nest_i(t), & \text{if } p_r < p_a \\ nest_i(t), & \text{otherwise} \end{cases}, \quad (12.6)$$

$$i = \{0, 1, 2, \dots, N_{nest} - 1\}$$

where $p_r \sim U(0, 1)$, p_a represents the replacement probability of $nest_i(t)$ and $p_a = 0.25$.

- **Mutation Operation**

To ensure diversity of solutions, nesting also requires a mutation process, i.e., each dimension of each nest may change its internal value. Mutation is defined as follows:

$$nest_i^j(t) = \begin{cases} 1 - nest_i^j(t), & \text{if } p_s < p_m \\ nest_i^j(t), & \text{otherwise} \end{cases}, \quad (12.7)$$

$$i = \{0, 1, 2, \dots, N_{nest} - 1\}, \quad j = \{0, 1, 2, \dots, N_{dime} - 1\}$$

where $p_s \sim U(0, 1)$, p_m represents the mutation probability of $nest_i^j(t)$ and $p_m = 0.01$.

12.3.3.2 Pareto-Based Multi-objective Optimization

- **Crowding Degree Calculation**

Nests in the front tier of the Pareto curve are superior to those in the back tier, and nests in the same tier will be ranked according to the degree of crowding, with the more crowded nests being superior. The crowding degree is defined as follows:

$$CD_i = (f_{1,i+1} - f_{1,i-1})/(f_{1,\max} - f_{1,\min}) + (f_{2,i+1} - f_{2,i-1})/(f_{2,\max} - f_{2,\min}), \quad (12.8)$$

where i represents $nest_i$ in one layer, $f_{1,\max}$ and $f_{1,\min}$ represent the maximum and minimum values of f_1 in this layer, respectively, and $f_{1,i+1}$ and $f_{2,i+1}$ are the fitness of the objective function f_1 and f_2 for $nest_{i+1}$ adjacent to $nest_i$, respectively.

- **Rapid Non-dominated Order**

We expect that the smaller the number of channels, the higher the detection rate, which reduces the time cost and improves the performance of our method. The following rule applies: if every objective function f_i in a $nest_i$ is smaller than another $nest_j$ (including f_1 and f_2) (both f_1 and f_2 in $nest_i$ (f_1, f_2) are smaller than $nest_j$ (f_1, f_2)),

We use two parameters S_p and N_p to record the dominant relationship of each nest. Here, S_p denotes a nest dominated by nest_p and N_p denotes a nest dominated by nest_p . The rapid non-dominated sorting algorithm is expressed as follows:

1. l_k denotes the number of layers of the Pareto curve, k represents the level of the stratum, $k=0$ for initialization, l_0 will consist of nest_p . l_0 will be composed of nest_p , where N_p is 0.
2. We traverse nest_q in S_p of l_k and perform $N_q = N_q - 1$; if N_q is zero, nest_q will belong to l_{k+1} .
3. $k = k + 1$.
4. Return to step (2) until all nests have been ranked.

12.3.4 Detector Structure and Training Strategies

Every nest represents a solution, and the fitness f_1 of each solution is computed based on the trained detectors. Therefore, detector training time limits the temporal complexity of our method. We approach the acceleration of detector training in terms of both a simple structure and a fast training strategy. Although a simple structure can save training time, an overly simple structure will not be able to meet the requirements of detection performance. We design two dense fully connected networks to validate the detector.

As a binary classification problem, the loss function of the detector is defined as follows:

$$\text{Loss}_D = - (y_{true} \times \log(y_p) + (1 - y_{true}) \times \log(1 - y_p)), \quad (12.9)$$

where y_p is the predicted value of the detector activated by the $\text{Sigmoid}(\cdot)$ function, $\log(\cdot)$ is the logarithmic function, and y_{true} is the ground truth.

For the training dataset, we first generate adversarial examples by attacking the classifier using different attacks. After that, we select the adversarial examples that can successfully spoof the classifiers and the clean examples that can predict correctly. Next, we perform multiple channel transformations on the selected examples. These transformed examples are fed into the classifier to generate the confidence matrix. The confidence matrices of the adversarial examples are labeled 1, while the confidence matrices of the clean examples are labeled 0. We extract 70% of all confidence matrices as the training set and the remaining as the test set. We train the detectors using an Adam optimizer with a learning rate of 0.0001 and a batch size of 16.

A fast detector training method based on randomized weight parameter sharing is applied in the CS process to accelerate the training process. This includes two aspects: weight parameter sharing to accelerate training, and parameter random crossover for different solutions. During the initialization process, the detectors corresponding to all initial solutions are trained for 10 ephemeral times to achieve a relatively high detection rate. The updated solutions have the following initial weights defined for the detectors:

$$w_i(t) = \frac{w_{j1}(t-1) + w_{j2}(t-1)}{2}, \quad (12.10)$$

where $w_i(t)$ represents the weight parameters of the detector corresponding to the i -th solution of the t -th iteration, and $w_{j1}(t-1)$ and $w_{j2}(t-1)$ are the weight parameters of the detectors randomly selected from the previous iteration.

12.4 Experiments and Analysis

Comprehensive experiments were carried out to verify the performance of our method, including the following: (1) a visual analysis of the channel transformations and (2) a comparison of the detection results.

12.4.1 Setup

Datasets: We evaluate the detection defensibility of ACT-Detector on the MNIST, CIFAR10, and ImageNet datasets.

DNNs: Different classifiers were adopted for the various datasets. We used three CNN-based classifiers for MNIST, namely CNN-MA, CNN-MB, and CNN-MC.

Attack methods: Several adversarial attack methods are applied to validate the detection performance of our method. The white-box attack includes FGSM [12], MI-FGSM [19], DeepFool [20], and C&W [21]. The black-box attack contains LSA [22], PWA [23], CRA [24], and GBA [25]. All of these attacks are based on the foolbox tool.

Defense baselines: In order to compare the detection results with our method, eight detection defenses were used, including the 45C-Detector [18], statistical detection (Sta-D) [26], perturbation detection (Per-D) [27], not twins detection (NT-D) [28], adversarial training-based detection (Advt-D) [29], ensemble model-based detection (Ens-D) [30], PGD-based detection (PGD-D) [31], and GAT [32].

Metrics: The metrics used in the experiment are as follows: perturbation 2-norm, attack success rate, detection rate, false positive rate, precision rate, recall rate, and miss rate.

12.4.2 Visual Analysis of Channel Transformations

In this section, we will show the visualization of the selected ACTs and further analyze the performance of the ACTs in terms of detection results compared to the remaining channels. After that, we will visualize and analyze the role of different channel transformations in the classifier based on Grad-CAM's attention mapping [33].

First, the visualization of the ACT selection process consists of visualizing the channel optimization results of our method on different datasets and different classifiers. We not only have to take into account the significant contribution of each channel to the detection results but also the large differences in channel types. Therefore, the final ACT selection during the optimization process is based on two criteria: channel contribution to the detection results and channel diversity. Figure 12.2 shows the ACTs selected on different classifiers of the MNIST dataset. We repeated the code 10 times and each time we recorded the near-optimal solution. Afterwards, we averaged all the near-optimal solutions to obtain the contribution of each channel transform to the detection results. In addition to the “clean” channels, we made sure to select one channel for each of the remaining channel types (resize, rotate, shift, and noise) to fulfill the channel selection criteria. As can be seen in Fig. 12.2, the ACTs selected

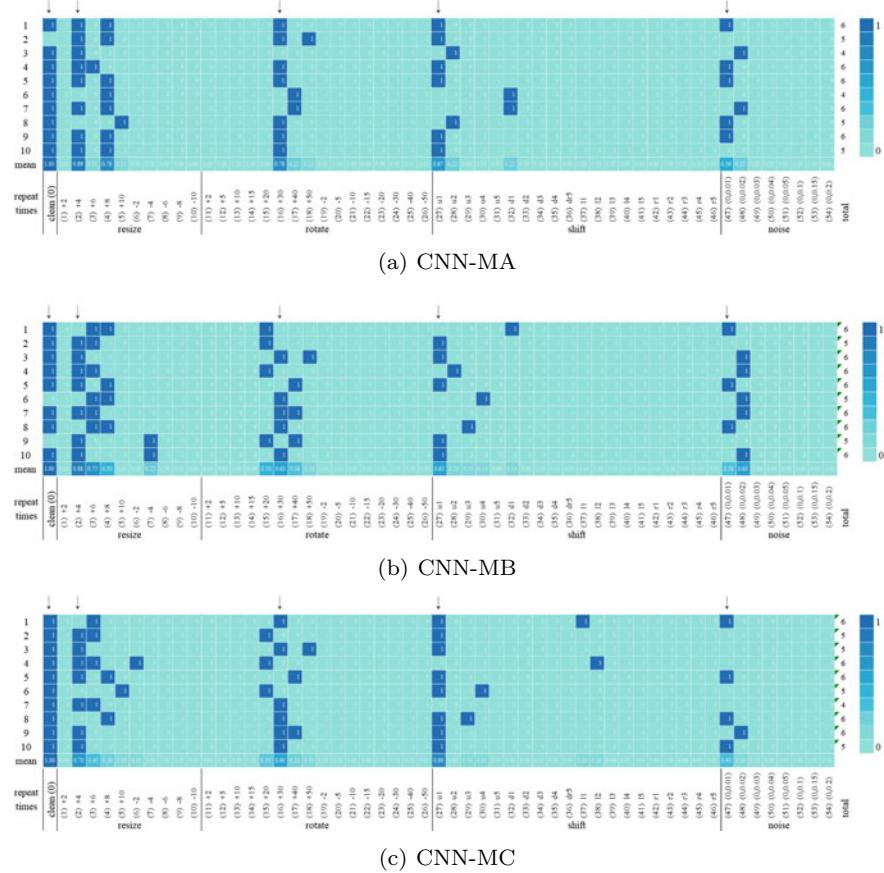


Fig. 12.2 The visualized heatmap of the ACT on the MNIST dataset. Red represents the final channel position selected, and green represents the unselected channel position. “↑” represents the selected channels

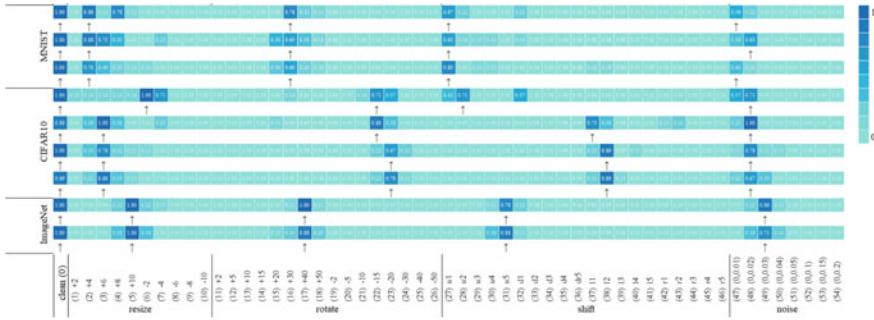


Fig. 12.3 The visualized heatmap of the selected ACT on different datasets. From top to bottom, each row represents the CNN-MA CNN-MB, and CNN-MC of MNIST, the ResNet32, ResNet56, VGG16, and VGG19 of CIFAR10, the ResNet-v2 and Inc-v3 of ImageNet

for the same classifiers are similar during each optimization, while the ACTs selected for different classifiers are slightly different. The “clean” channel is almost always selected for each optimization. This is because the essence of the detection method lies in the difference between the antagonistic and clean examples in the different channel transformations. Visualization of the ACT optimization process for different classifiers on the CIFAR10 and ImageNet datasets.

Figure 12.3 shows the adaptive ACT heatmap of various classifiers on different datasets.

It can be concluded that for different datasets, better adversarial detection can be obtained by choosing adaptive channels, i.e., different types and number of channels. This is difficult to decide by human experience, while it is much easier by BCS optimization. From top to bottom, the rows represent CNN-MA, CNN-MB, and CNN-MC for MNIST; ResNet32, ResNet56, VGG16, and VGG19 for CIFAR10; and ResNet-v2 and Inc-v3 for ImageNet. Each line represents the average result after 10 repetitions of encoding, and “↑” indicates the selected ACT. As shown in Fig. 12.3, we ended up selecting five channel transforms for each dataset for each classifier, thus reducing the spatial complexity of the detectors. In addition, we find that the selected ACTs are similar across datasets. Take the “rotation” transform as an example: On the MNIST dataset, choosing different classifiers results in “rotation +30°”. On the CIFAR10 dataset, the choice results in “rotate -15°” or “rotate -20°”, which are very close. On the ImageNet dataset, the selection results are both “rotate +40°”. As the image size increases, the absolute value of the “rotate” angle also increases. This is because, for large image sizes, a larger rotation angle is needed to remove the perturbations in the confrontation example.

Then, in order to demonstrate the effectiveness of the selected ACT, we compare the detection performance of the ACT with the remaining channels. The rest of the channels are defined as supplementary channels and are denoted as “50C”. We randomly select five channels from the supplementary channels and repeat the selection three times, which are denoted as “5C-1”, “5C-2”, and “5C-3”. The detector is trained according to “ACT”, “50C”, “5C-1”, “5C-2”, and “5C-3”. Figure 12.4 lists

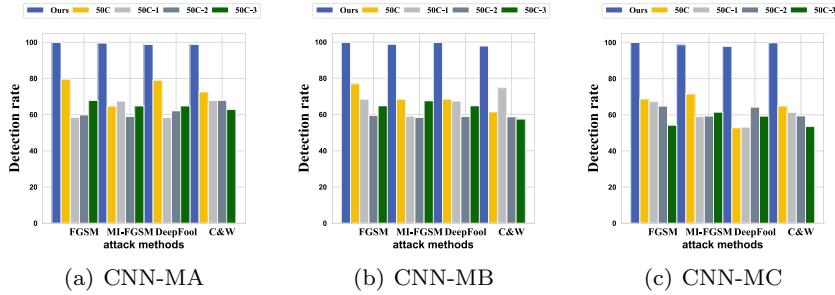


Fig. 12.4 The detection rate comparison between the ACT of our method and the complement channels on different classifiers of the MNIST dataset

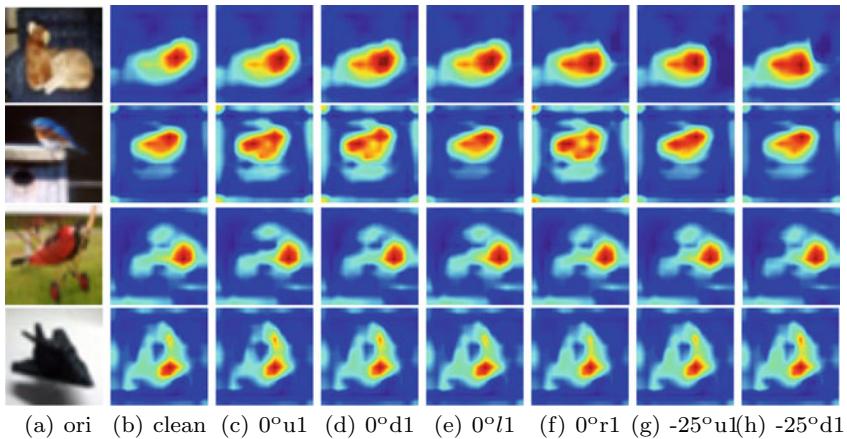


Fig. 12.5 The attention maps of different channel transformations. The visualization method is Grad-CAM [33]. The channel transformations are randomly selected from the 45 channels of the 45C-Detector. The red region represents the attention region of the classifier. The change in the size and position of the red region demonstrates the difference in attention features

the comparison of the detection rates of different detectors. The ACT detector can effectively detect adversarial examples with a detection rate close to 100.00. Although the detection rate of the “50C”-based detector decreases significantly, it is still higher than that of the “5C-1”, “5C-2”, and “5C-3”-based detectors. These results indicate that among the candidate channels, the contribution of the ACT to the detection results is much larger than that of the complementary channels, which suggests that the selected ACT is compact and effective.

Lastly, we visually analyze the role of different channel transformations in the detector. Using the ResNet-v2 classifier on the ImageNet dataset as an example, we used the Grad-CAM method to generate the attention maps of different channel transformations acting on different images, as shown in Figs. 12.5 and 12.6 are shown. Within each figure, the first column represents a different original

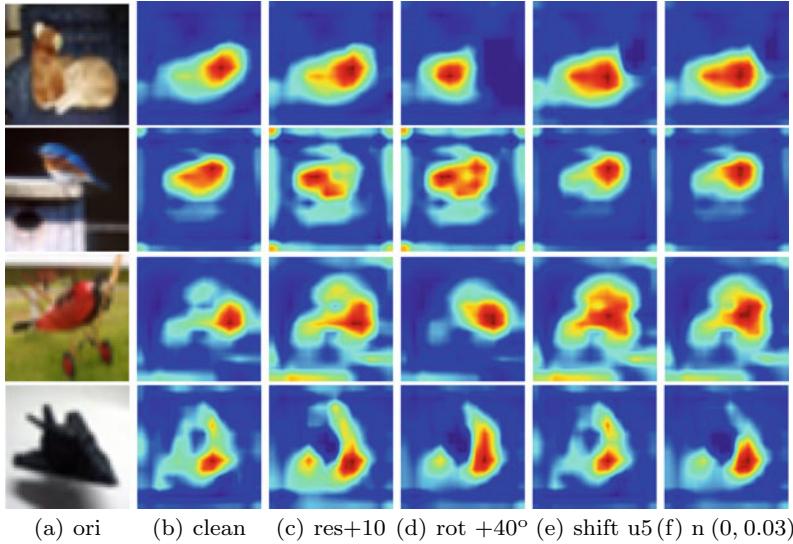


Fig. 12.6 The attention maps of different channel transformations. The visualization method is Grad-CAM [33]. The channel transformations are selected by our method

image, and the subsequent columns represent the visualization of the attention maps after the images have been transformed by different channels. Areas in red indicate the attention regions of the classifier. Variations in the size and position of the red regions indicate differences in the features to which the classifier pays attention. We argue that the detector detects adversarial examples and pure examples by focusing on the differences in features across different channel transformations. Figure 12.5 shows the attention maps generated for seven randomly selected channels from the 45C-Detector. As can be seen, the attention maps of these channels are very close to each other, leading to feature redundancy between the channels. Figure 12.6 shows the attention maps of the near-optimal channels selected by the ACT detector. It is obvious that there are significant differences in the attention maps between channels, which is the reason why our method can use as few channels as possible to achieve a high detection rate. From this, it can be inferred that although the number of channels selected by the ACT detector is small, it already contains the key features needed to detect adversarial examples.

12.4.3 Comparison of Detection Results

In this section, we discuss how well our method detects malicious and clean examples and compare them to a baseline. After that, we will analyze the effectiveness of our method's detection against white-box and black-box attacks.

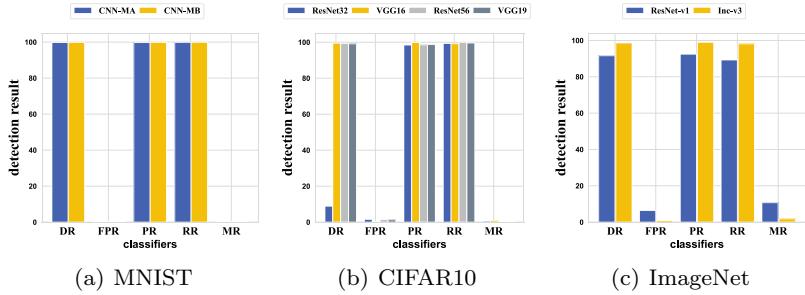


Fig. 12.7 The detection results against the adversarial and clean examples for different classifiers on different datasets

First, we use our method's selected ACTs to detect malicious and clean examples and analyze the detection results based on different metrics, as shown in Fig. 12.7. Since these three models perform very similarly on the MNIST dataset, we only show the results for CNN-MA and CNN-MB in later experiments. Due to the use of eight attacks, the number of adversarial examples is much larger than the number of clean examples, which leads to sample imbalance during training. Therefore, we mixed all the adversarial examples and randomly selected adversarial examples equal to the number of clean examples. The randomly selected adversarial examples were mixed with the clean examples. We randomly selected 70% of them as the training set and the remaining 30% as the test set. Taking the ImageNet dataset as an example, 5,000 clean examples were used in the experiments, and 37,621 adversarial examples were obtained against the ResNet-v2 model using different attacks. From the 37,621 examples, 5,000 adversarial examples are randomly selected and mixed with 5,000 clean examples to obtain 10,000 examples. Of these, 7,000 were used as the training set and 3,000 as the test set.

Based on the detection results in Fig. 12.7, our method has a high detection rate of nearly 100.00% on the MNIST and CIFAR10 datasets. In the ImageNet dataset, the detection rate decreases slightly. This is mainly due to the larger image size of the ImageNet dataset, which results in smaller differences between the various channel transforms. Effective detection results of our method are not only reflected in DR metrics but also in other metrics. FPR is the proportion of clean examples incorrectly detected as antagonistic examples. When an excess of clean examples are incorrectly detected as adversarial examples, the ability of the classifier to handle normal tasks will be seriously affected. Our method's FPR metric is very low, indicating that the normal tasks of the classifier will not be affected too much. In particular, our method also exhibits high PR and RR values, as well as low MR values, reflecting its effective detection results. Figure 12.8 shows the corresponding confusion matrices for different datasets and different classifiers in the detection of the adversarial example and clean example tasks. From the figure, it can be seen that the values on the diagonal are large, indicating satisfactory detection results.

Figure 12.9 lists the comparison of the average detection rates of our method and baselines on different datasets. The “Mean” and “StDev” denote the mean and

	adv	clean		adv	clean		adv	clean		adv	clean
adv	20,875	55	adv	20,881	33	adv	17,860	280	adv	17,973	245
clean	30	21,040	clean	24	21,062	clean	120	17,740	clean	7	17,755
(a) CNN-MA			(b) CNN-MB			(c) ResNet32			(d) ResNet56		
(e) VGG16			(f) VGG19			(g) ResNet-v2			(h) Inc-v3		

Fig. 12.8 The confusion matrix for detection against adversarial and clean examples

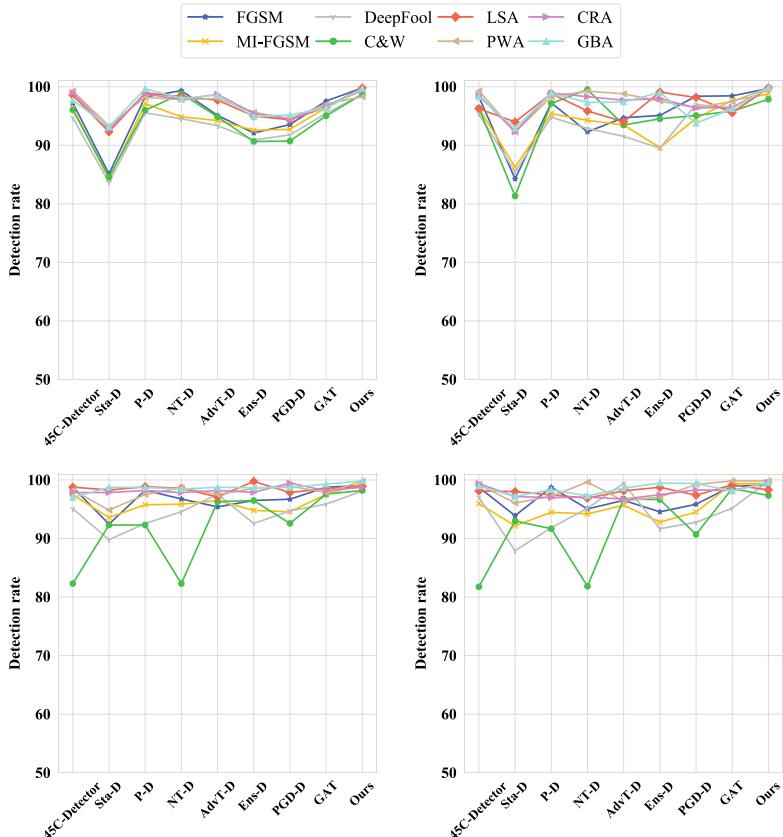
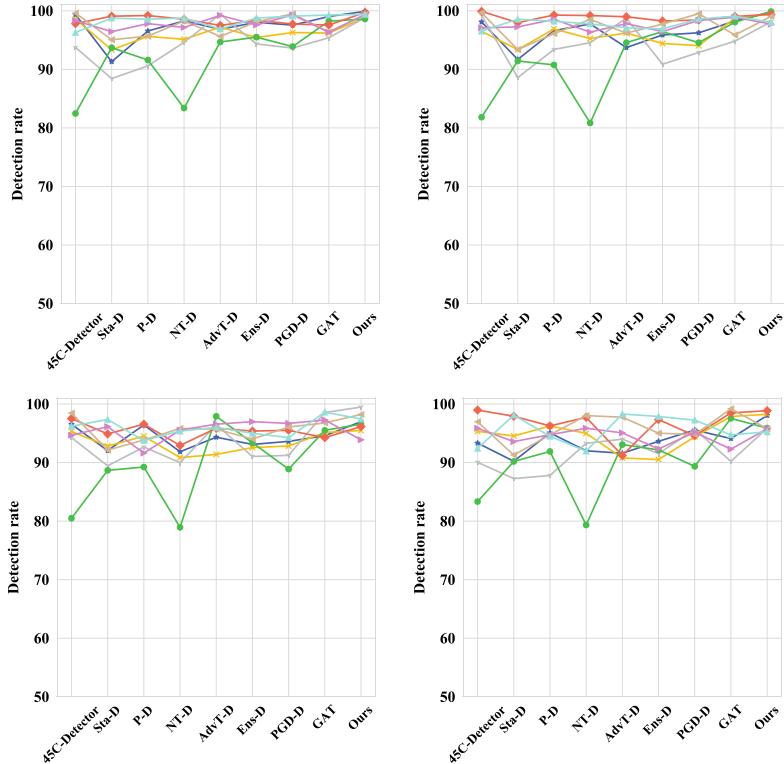


Fig. 12.9 The mean detection rate comparison between our method and baselines against the adversarial and clean examples on different datasets

**Fig. 12.9** (continued)

standard deviation of the detection rates of the counter-examples produced by the defense method against different attacks. The average detection rate measures the effectiveness of the detection, and the standard deviation measures the stability of the detection performance. It can be seen that the average detection rate of our method is the highest on each dataset for each classifier. Our method also has the smallest standard deviation of the detection rate, which indicates that its detection performance is more stable compared to the baseline. Therefore, our method can provide effective detection results for classifiers with different structures and datasets with different data volumes.

12.5 Conclusions

Deep learning is vulnerable to adversarial examples. We have designed a channel optimization technique based on confidence differences to defend against adversarial attacks. We demonstrate the following in our experiments: (1) that our adversarial example detection strategy is able to proactively defend against and detect most adversarial examples. (2) The adaptive channel optimization is applicable to the classification of white-box and black-box attacks. (3) With ACT-Detector, the complexity of detector training is effectively reduced. However, there is still room for improvement in attack-type classification. In the future, we will further study the different responses to various attacks to distinguish them in more detail.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 770–778. IEEE Computer Society (2016)
2. Chen, X., Yuan, G., Wang, W., Nie, F., Chang, X., Huang, J.Z.: Local adaptive projection framework for feature selection of labeled and unlabeled data. *IEEE Trans. Neural Netw. Learn. Syst.* **29**(12), 6362–6373 (2018)
3. Chen, J., Zheng, H., Xiong, H., Wu, Y., Lin, X., Ying, S., Xuan, Q.: Dgepn-gcen2v: a new framework for mining ggi and its application in biomarker detection. *SCIENCE CHINA Inf. Sci.* **62**(9), 1–3 (2019)
4. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a Meeting Held December 3–6, 2012, Lake Tahoe, Nevada, United States, pp. 1106–1114 (2012)
5. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–14 (2015)
6. Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **39**(6), 1137–1149 (2017)
7. Chen, J., Zheng, H., Xiong, H., Shen, S., Su, M.: Mag-gan: massive attack generator via gan. *Inf. Sci.* **536**, 67–90 (2020)
8. Shang-Tse, C., Cory, C., Jason, M., Duen Horng, C.: Shapeshifter: robust physical adversarial attack on faster r-cnn object detector. In: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10–14, 2018. Lecture Notes in Computer Science, vol. 11051, pp. 52–68. Springer (2018)
9. Fionda, V., Pirro, G.: Community deception or: How to stop fearing community detection algorithms. *IEEE Trans. Knowl. Data Eng.* **30**(4), 660–673 (2018)
10. Akhtar, N., Mian, A.: Threat of adversarial attacks on deep learning in computer vision: a survey. *IEEE Access* **6**, 14410–14430 (2018)
11. Chen, J., Zheng, H., Chen, R., Xiong, H.: Rea-soc: a novel adversarial defense by refocusing on critical areas and strengthening object contours. *Comput. Secur.* **96**, 101916.1–10916.18 (2020)
12. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–10 (2015)

13. Papernot, N., McDaniel, P., Wu, X., Jha, S., Swami, A.: Distillation as a defense to adversarial perturbations against deep neural networks. In: IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22–26, 2016, pp. 582–597. IEEE Computer Society (2016). <https://doi.org/10.1109/SP.2016.41>
14. Jin, G., Shen, S., Zhang, D., Dai, F., Zhang, Y.: Ape-gan: adversarial perturbation elimination with gan. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019, Brighton, United Kingdom, May 12–17, 2019, pp. 3842–3846. IEEE (2019)
15. Sankaranarayanan, S., Jain, A., Chellappa, R., Lim, S.N.: Regularizing deep networks using efficient layerwise adversarial training. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, pp. 4008–4015. AAAI Press (2018)
16. Dongyu, M., Hao, C.: Magnet: a two-pronged defense against adversarial examples. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30–November 03, 2017, pp. 135–147. ACM (2017)
17. Mustafa, A., Khan, S.H., Hayat, M., Shen, J., Shao, L.: Image super-resolution as a defense against adversarial attacks. *IEEE Trans. Image Process.* **29**, 1711–1724 (2020)
18. Tian, S., Yang, G., Cai, Y.: Detecting adversarial examples through image transformation. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2–7, 2018, pp. 4139–4146. AAAI Press (2018)
19. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp. 9185–9193. IEEE Computer Society (2018)
20. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 2574–2582. IEEE Computer Society (2016)
21. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22–26, 2017, pp. 39–57. IEEE Computer Society (2017)
22. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Local search attack in foolbox tool. <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks/score.html#foolbox.attacks.LocalSearchAttack>
23. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Point wise attack in foolbox tool. <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks/decision.html#foolbox.attacks.PointwiseAttack>
24. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Contrast reduction attack in foolbox tool. <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks/decision.html#foolbox.attacks.ContrastReductionAttack>
25. Jonas, R., Wieland, B., Behar, V., Evgenia, R.: Gaussian blur attack in foolbox tool. <https://foolbox.readthedocs.io/en/v1.8.0/modules/attacks/decision.html#foolbox.attacks.GaussianBlurAttack>
26. Grosse, K., Manoharan, P., Papernot, N., Backes, M., McDaniel, P.: On the (statistical) detection of adversarial examples. ArXiv Preprint, pp. 1–8 (2017)
27. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings, pp. 1–12. OpenReview.net (2017)
28. Gong, Z., Wang, W., Ku, W.S.: Adversarial and clean data are not twins. ArXiv Preprint, pp. 1–7 (2017)
29. Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., McDaniel, P.: Ensemble adversarial training: Attacks and defenses. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings, pp. 1–20. OpenReview.net (2018)

30. Ju, C., Bibaut, A., van der Laan, M.: The relative performance of ensemble methods with deep convolutional neural networks for image classification. *J. Appl. Stat.* **45**(15), 2800–2818 (2018)
31. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–28. OpenReview.net (2018)
32. Yin, X., Kolouri, S., Rohde, G.K.: Gat: Generative adversarial training for adversarial example detection and robust classification. In: Proceedings of the Eighth International Conference on Learning Representations, pp. 1–26 (2020)
33. Selvaraju, R.R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., Batra, D.: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22–29, 2017, pp. 618–626. IEEE Computer Society (2017)

Chapter 13

Defense Against Free-Rider Attack from the Weight Evolving Frequency



13.1 Introduction

Federated learning (FL) [1–4] has been proposed as a style of distributed method [5, 6] in machine learning where a global model is trained and clients update local model parameters, like gradient, with no sharing of their private data and partitioning them. Taking into account the remarkable advantages in privacy preserving, the FL is already applied to different kinds of data mining, e.g., credit prediction [7, 8], wellness evaluation [9, 10], and the next word prediction [11, 12].

In a conventional FL system, every client is required to contribute its own data for global model training. In return, as a reward, the client is entitled to use the finally train global model. That results in a free-rider attack [13, 14]. In such cases, the high-value global model is available to clients that do not contribute anything. Such clients are termed as “free-riders”. Generally, in a shared resource environment there is always a “free-rider” problem, i.e., a “free-rider” benefits from the environment without contributing anything.

In this chapter, we investigate the “hitchhiker” attack in FL systems. It is noticed that almost several existing efforts are dedicated to address the “hitchhiker” problem in FL, mainly in two aspects, the outlier detection [13] for model parameters and the customer’s contribution evaluation [15, 16].

Existing approaches used to defend against free-rider attacks remain challenged in three areas, i.e., (1) what is the best way to defend against advanced camouflaged free-riders, (2) what is the most appropriate way to cope with scenarios in which multiple free-riders are present (more than 50% of the clients), and (3) what is the optimal way to balance main-task performance with the effectiveness of the defenses. We reconsider the differences between benign customers and free-riders during our dynamic training sessions in order to overcome these challenges. Amazingly, we have observed that free-riders can exploit the global model aggregated and distributed by the server to artificialize its model weights, similarly to benign clients.

However, they cannot disguise that the model weights are optimized. This is because “free-riders” will not train normally, and hence, they cannot efficiently evade

as well as their benign neighbors. Hence, it is an intuitive consideration to identify free-riders utilizing model evolutionary information. Instead, we have defined a frequency of evolution of model weights, that is a value of a statistic that involves no proprietary information, to measure the difference between “hitchhikers” and “benign customers”, which are recorded as having drastically varying values of model weights.

We observe the difference between free-riders and benign clients during FL training, and inspired by this, we present a judgmental way of defense approach based on the frequency of weight evolutions. In particular, we define the concept of Weight Evolution Frequency Matrix (WEF-Matrix), used to record the frequency of evolution of weights in the model’s penultimate layer. Our method computes the variation of the weights between two consecutive rotations of the local training and assesses the evolution frequency of all the weights by averaging the range of the overall variations as a dynamic threshold. The WEF-Matrix of the local model needs to be uploaded to the server by every client along with the model weights. A server can then separate free-riders or benign clients based on the Euclidean distance, cosine similarity, and the overall average frequency of the WEF-Matrix between clients. For both benign clients and free-riders, the server will aggregate and distribute different global models based only on their evolving frequency differences. Thus, the global models obtained by free-riders do not have the model weights contributed by benign clients, which prevents free-riders from stealing trained high-value models.

The main contributions of this chapter are summarized as follows.

- We initially observe that the information about the dynamics of benign customers and free-riders is different over the course of local training in FL. We have emphasized the potential for detecting free-riders by using the frequency of evolution of the model repertoire during the training process.
- We are inspired to propose a new method. A WEF-Matrix is designed to collect the frequency of changes in model weights during the whole training process of each client, and we use it as an effective means of detecting hitchhikers.
- For situations where the main clients are free-riders (i.e., 50% or even up to 90%), a personalized model aggregation strategy is used by our method to defend against attacks in the early training phase.
- An extensive experimentation was conducted on five datasets and five models. Results indicate that our method obtains much better defense than the state-of-the-art (SOTA) baseline and identifies free-riders in the early stages of training. Moreover, it is also effective in preventing adaptive attacks. Further, we present weight visualizations to explain its effectiveness.

13.2 Related Work

In this section, we review the related work and briefly summarize attack and defense methods used as baselines in the experiments.

13.2.1 Free-Rider Attacks on Federated Learning

According to the attacker's camouflage tactics, free-rider attack includes ordinary attacks [13], random weight attack [13], stochastic perturbations attack [17], and delta weight attack [13].

A normal attack [13] is a plain attack without camouflage, and the malicious client does not have access to any local data, i.e., it is not trained locally. It has access to the server's published global model by participating in FL training. Building on this, a random weight attack [13] is used to construct a gradient estimation matrix by randomly sampling each value from a uniform distribution with a given range $[-R, R]$. Nevertheless, this works well only under the condition that the range values R are ideally chosen in advance. Moreover, weights that are randomly generated generally cannot guarantee the favorable attack performance by emulating the modeled weights of benign clients. Moreover, weights that are randomly generated [17] generally cannot guarantee the favorable attack performance by emulating the modeled weights of benign clients. In comparison to the previous attacks, the delta weight attack (DWA) delivers a carefully crafted update to the server by computation of the difference between the weights of the last two rounds. It should be noted that in machine learning training, weight variation in each round is small except for the first few epochs. Therefore, faked updates may be similar to updates from benign clients.

13.2.2 Defenses Against Free-Rider Attacks

The existing defense methods can be mainly categorized into two types, i.e., outlier detection of model parameters and clients' contribution evaluation.

In the first research on free-rider attacks on FL, a possible defense approach based on detection of outlier points, designated STD-DAGMM, explored by Jierui et al. [13], according to which a standard deviation metric, [18], was added on top of the depth-autocoded Gaussian mixture model. The network architecture is categorized into two parts: the compression network and the estimation network. In particular, gradient renewal matrices are fed to the compression net to obtain low-dimensional output vectors, while the estimation network takes the standard deviation of the input vectors and enters it into the estimation network. Standard deviations from the input variables are computed and then vectorially stacked with the compiled Euclidean and cosine distance metrics. Eventually, this voltage is linked in a series with the learned low-dimensional vector of representations of the compression network. Connected output vectors are then fed to the estimation network for a multivariate Gaussian estimation. Also, it is difficult to choose an appropriate threshold to separate the free-riders from the benign clients when the free-riders account for more than 20% of the total number of clients.

Another approach to the defense of free-rider attacks is to evaluate the clients' responses based on their contributions. Lyu et al. [19] proposed a Collaborative

Fairness Federated Learning (CFFL) that achieves fairness in cooperation through a reputation mechanism. It primarily evaluates the respective contribution of every client by exploiting the server's validation dataset. Each client iteratively renews its individual rating, while the server assigns models with different qualities based on their contribution. Clients with higher reputations receive better quality aggregated models. Nevertheless, CFFL relies on proxy datasets, which is not practical in real-world applications. With this foundation, Xinyi et al. suggested Robust Fairness Federated Learning (RFFL), to achieve collaborative fairness and adversarial robustness through a mechanism of reputation. In RFFL, the speaker iteratively evaluates the respective contribution of each client by the cosine of similarity between the uploaded local gradient and the aggregated global gradient. Compared to CFFL, RFFL eliminates the need to build a validation dataset beforehand. Nevertheless, when faced with adaptive "free-riders" that have the ability to camouflage the gradients under non-ID data, RFFL does not work.

13.3 Methodology

13.3.1 Overview

The notion of sensitive neurons has recently been extensively discussed (citeX-uPZ20,MalmiercaNNPPE19). When data is fed into a neural network, for example, it has been observed that not all neurons are activated. The weights will vary considerably, as various neurons in different layers will react to different strengths of the data features. Free-riders do not have data, therefore when they produce fake updates they do not have the necessary information to take into account how both sensitive and insensitive neurons affect the parameters. Thus, it is harder for free-riders to pseudo-update the frequency of weight changes. Motivated by this, our method adopts the weight change frequency during partial model training as an effective means of defending against hitchhikers. The overview of our method is shown in Fig. 13.1 and includes three main parts: ① WEF-Matrix information collection (Sect. 13.3.2), ② client separation (Sect. 13.3.3), and ③ personalized model aggregation (Sect. 13.3.4).

13.3.2 WEF-Matrix Information Collection

In order to obtain valid information about our customers, WEF-Matrix collects it in three steps: (i) WEF-Matrix initialization, (ii) Threshold determination, and (iii) WEF-Matrix calculation.

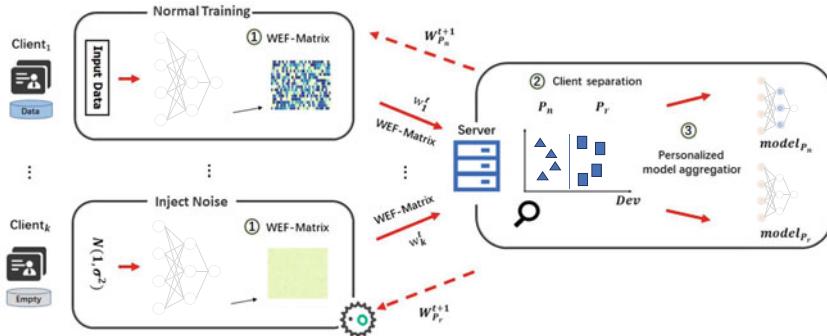


Fig. 13.1 Overview of our method. The client uses the initialized WEF-Matrix to record the evolution frequency of the selected layer weight in the local training epoch. WEF-Matrix is then sent to the server along with the model updates. According to the WEF-Matrix, the server separates benign clients and free-riders to form $\{P_n, P_r\}$. The model updates uploaded by the two sets of clients are aggregated separately and delivered to the clients in respective sets only

13.3.2.1 WEF-Matrix Initialization

We define firstly the WEF-Matrix, decided by the client p_i weights of the penultimate layer $w_{i,s} \in \mathbb{R}^{H \times W}$ and initialized to an all-zero matrix. It registers information on the frequency of weight progression in the local training. The reason we use the weights of the penultimate layer is for the following reasons. A final classification result is achieved by the softmax output of the last layer. There is a greater impact on the final classification result the closer the weights are to the last layer, which is more representative of the weight changes in this layer. The initialization process is as follows:

$$F_i^0 = \text{zeros}(H, W) \quad (13.1)$$

where $\text{zeros}(m)$ returns an all-zero matrix of size $H \times W$. F_i^0 has the same size as $w_{i,s}$.

13.3.2.2 Threshold Determination

During the local training process on the client side, we collect the corresponding weight change frequencies through the initialized WEF-Matrix. We need to determine a dynamic threshold for measuring the frequency change before computing the WEF-Matrix. Assume that the client p_i is undergoing the $(t' + 1)$ -th round of local training, and the model weights obtained after training are $w_i^{t'+1}$. The weight of the client p_i in the penultimate layer, denoted as $w_{i,s}^{t'+1}$, is chosen. However, we calculate the change in weight between $w_{i,s}^{t'+1}$ in the $(t' + 1)$ round and $w_{i,s}^{t'}$ in the t' round, and take the

overall average change as the threshold. Calculate the threshold of client p_i at the $(t'+1)$ -th round as follows:

$$\alpha_i^{t'+1} = \frac{\sum_{j=1}^H \sum_{k=1}^W |w_{i,s,j,k}^{t'+1} - w_{i,s,j,k}^{t'}|}{H \times W} \quad (13.2)$$

where $w_{i,s,j,k}^{t'+1}$ is a weight value of the j -th row, $|\cdot|$ returns the absolute value and the k -th column from the penultimate layer of the client p_i in the $(t'+1)$ -th round, H and W represent the rows and columns of $w_{i,s}^{t'+1}$, respectively.

13.3.2.3 WEF-Matrix Calculation

We formulate the frequency of weight evolution in local training based on the calculated dynamic threshold. The computational procedure is as follows:

$$F_{i,j,k}^{t'+1} = \begin{cases} F_{i,j,k}^{t'} + 1, & |w_{i,s,j,k}^{t'+1} - w_{i,s,j,k}^{t'}| > \alpha_i^{t'+1} \\ F_{i,j,k}^{t'}, & otherwise \end{cases} \quad (13.3)$$

where the k -th column of the client p_i in the $(t'+1)$ -th round, $j = \{1, 2, \dots, H\}$, $k = \{1, 2, \dots, W\}$, and $F_{i,j,k}^{t'+1}$ represents a frequency value of the j -th row. The amount of frequencies calculated in each round will be accumulated. At the end, this WEF-Matrix is uploaded to the server by the client along with the model updates. It is worth noting the uploaded information does not involve the client's data privacy.

13.3.3 Client Separation

To tell benign customers from free-riders, there are three metrics that we compute using the difference in WEF-Matrix and combine them to detect free-riders. The server randomly selects a client p_i , then based on its uploaded WEF-Matrix, calculates (1) the Euclidean distance Dis , (2) the cosine similarity Cos with other clients' WEF-Matrix, and (3) the average frequency Avg of their WEF-Matrix, as follows:

$$Dis_i = \sqrt{\sum_{j \in K} (F_i - F_j)^2}, \quad i \neq j \quad (13.4)$$

where K represents the total number of clients and F_i represents the WEF-Matrix uploaded by the client p_i .

$$Cos_i = \frac{F_i \cdot F_j}{\|F_i\| \|F_j\|} \quad (13.5)$$

where $\|\cdot\|$ represents the 2-norm of the matrix and \cdot represents the matrix dot product.

$$Avg_i = \frac{\sum_{j=1}^H \sum_{k=1}^W F_{i,j,k}}{H \times W} \quad (13.6)$$

where H and W represent the rows and columns of F_i , respectively.

For the customer p_i , we add the standardized deviations ΔDis , ΔCos , as well as ΔAvg to further compute the value of the similarity deviation Dev , as follows:

$$Dev_i = \frac{\Delta Dis_i}{\sum_{j=1}^K (\Delta Dis_j)} + \frac{\Delta Cos_i}{\sum_{j=1}^K (\Delta Cos_j)} + \frac{\Delta Avg_i}{\sum_{j=1}^K (\Delta Avg_j)} \quad (13.7)$$

By using three metrics for Dev , the reason for using three metrics for Dev is to fully account for the variety of scenarios in which free-riders may exist, as well as to reduce the rate of success of free-riders in bypassing defenses. Specifically, Euclidean distance can effectively be used to identify free-riders; however, due to its symmetry, when the number of benign customers is close to the number of free-riders, Euclidean distance is not useful. Hence, we exploit cosine similarity and average frequency to make a better distinction. Those three metrics work together in a complementary way.

The server sets the reputational threshold $/xi$ based on the similarity deviation value, and then divides the benign customers and free-riders into $\{P_n, P_r\}$. From the experimental evaluation, we find that the similarity bias gap between benign customers and free-riders is large, while the similarity bias gap between free-riders is small. Hence, a definite margin can be set to identify free-riders based on the largest similarity deviation value. We have set ξ in our experiments! = $\max(Dev) - \epsilon$ as the trust threshold. Among the hyperparameter ϵ has an adjustable range of values, so we set it by an initial study based on a small dataset and realized that this setting $\epsilon = 0.05$ is effective in average cases.

13.3.4 Personalized Model Aggregation

On the basis of the client separation process, it is possible for the server to uphold two separated models in each round and to aggregate the uploaded model updates from both groups of clients separately. Using the two groups $\{P_n, P_r\}$, the separator forms two global models, which are then distributed to the corresponding groups separately. Therefore, those global models trained by benign clients cannot be accessed by free-riders. The aggregation process is as follows:

$$\{P_n\} : w_g^{t+1} = w_g^t + \frac{1}{\sum \{P_n\}} \sum_{i \in P_n} (w_i^{t+1} - w_g^t) \quad (13.8)$$

$$\{P_r\} : w_g^{t+1} = w_g^t + \frac{1}{\sum \{P_r\}} \sum_{i \in P_r} (w_i^{t+1} - w_g^t) \quad (13.9)$$

where w_g^{t+1} and w_g^t are the global model in the $(t+1)$ -th round and the t -th round, w_i^{t+1} is the local model updates uploaded by the client. $\sum \{P_n\}$ and $\sum \{P_r\}$ represent the number of clients in their groups, respectively.

The client is separated by the server at each round. Evolution frequency of the weights of the model gathered by the server will be accumulated. Then, the variation in the evolution frequency of updates between benign clients and free-riders will further expand.

13.3.5 Algorithm Complexity

We present our complexity analysis of our method in two parts, i.e., information collection at the client and identification at the server. On the client side, we choose the weights of the penultimate layer of the model to initialize the WEF-Matrix, which is then used to record weight evolution frequency information. The computational complexity can be defined as

$$T_{client} \sim \mathcal{O}(1) + \mathcal{O}(T') \quad (13.10)$$

where T' is the local training epochs.

On the server, we compute Dev and execute model aggregation for the client in $\{P_n, P_r\}$, respectively. Therefore, the time complexity is

$$T_{server} \sim \mathcal{O}(K) + \mathcal{O}(K) \quad (13.11)$$

where K is the number of clients.

13.4 Experiments Setting

Datasets: We conducted an evaluation of our method on five datasets (i.e., MNIST, CIFAR-10, GTSRB, BANK, and ADULT). The MNIST citelecun1998gradient dataset includes 70,000 real-world handwritten images with numbers ranging from 0 to 9. The CIFAR-10 citekrizhevsky2009learning dataset includes 60,000 color images in 10 categories, 6,000 in each category, with a size of 32×32 . The GTSRB [20] dataset represents 51,839 real-world German transportation signs in color from 43 categories. There are 48,843 records in the ADULT [21] dataset. We have balanced the ADULT dataset manually so that it has 11,687 records above 50K and 11,687 records below 50K, resulting in a total of 23,374 records. The BANK

dataset is related to the activities of a direct sales campaign of a single banking institution in Portugal and contains the data on 45,211 customers whether or not they subscribe to a time deposit, with 16 attributes for each customer. For each dataset, an 80–20 train–test split was performed.

Number of clients: In all experimental scenarios, our main focus is to evaluate the influence of different percentages of free-riders on our approach. Hence, the total number of clients is 10, while we discuss free-rider attacks with 10%, 30%, 50%, and 90% of the total number of clients.

Models: Different classifiers are used for different datasets. For example, for MNIST, LeNet [22] was used for classification. For the more complex image datasets, CIFAR-10 and GTSRB, VGG16 [23] and ResNet18 [24] were used, respectively. MLP [25] was used for the structured datasets ADULT and ADULT BANK. Results of all evaluations are averages of three runs with the same settings.

Hyper-Parameters: For all experiments, we set the hyperparameters $\epsilon = 0.05$.

Attack Methods: In this chapter, we applied three existing “free-rider” attack methods to evaluate the detection performance, namely, random weight attack, random perturbation attack, and triangular weight attack. Of these, the random weight attack uses 10^{-3} for the weight generation range R . Within the adaptive attack scenario, we design a new free-rider attack to evaluate the defense performance.

Baselines: Comparisons were made using two defense methods, including CFFL [19] based on the validation dataset and RFFL [26] based on the cosine similarity between the local gradient and the aggregated global gradient. Undefended FedAvg aggregation algorithm /citemcmahan2017communication serves as a benchmark.

Evaluation Metrics: We evaluate the detection method’s performance by assessing the Highest Mean Precision (HMC) of the models which can be stolen by hitchhikers. The lower the HMC, the better the detection.

13.5 Evaluation and Analysis

In this section, we evaluate the performance of our method by answering the following five research questions (RQs):

- **RQ1:** In defending against various hitchhiking attacks, does our method achieve SOTA defense performance compared to the baseline?
- **RQ2:** Does our method still achieve optimal performance with a different percentage of free-riders in FL?
- **RQ3:** Does our method affect the execution of the main tasks? What is its communication overhead?

13.5.1 RQ1: Defense Effectiveness of Our Method

In this section, we validate the defense effectiveness of our method compared to the baseline on different datasets and models.

Implementation Details. (1) There were five datasets tested in both IID data and non-IID data settings. Problems of heterogeneity were explored for the non-IID data using a Dirichlet distribution with a default distribution coefficient of 0.5. (2) The number of free-riders is generally less than the number of benign clients. As a result, there are two scenarios with 10% and 30% free-riders among 10 clients, in which the free-riders are camouflaged by random weight attack (Rw), random perturbation attack (Sp), and delta weight attack (Dw), respectively. (3) Three baselines were used for our comparison, namely undefended FedAvg aggregation [1], RFLL [26], and CFFL [19]. An evaluation metric is the HMC available to free-riders. The results for IID data and non-IID data are shown in Figs. 13.2 and 13.3, respectively.

Results and Analysis. The experiments under IID and non-IID data distributions show that the overall defense performance of our method is the best compared to CFFL and RFLL in most cases. In particular, when dealing with the anisotropic nature of the non-IID data, the defense performance remains stable. Among the test results on three image datasets (i.e., MNIST, CIFAR-10, and GTSRB), the overall HMC obtained by the free-rider is no more than 37% as a result of our method, with the lowest HMC being only 3.52%. Results of the testing under the two textual datasets (i.e., ADULT and BANK) allowed the free-riders to obtain an overall HMC of no more than 70%, with the lowest HMC being only 35.22%. Remarkably, as both ADULT and BANK datasets have only two categories, models should have an accuracy of around 50% during initial training (i.e., random guessing). In summary, our method achieves SOTA performance.

Our method shows a stable defense performance when defending against different “free-rider” attacks. On the contrary, CFFL can be observed to have a defense performance that is not as stable as expected. For instance, as shown in Fig. 13.2, on the MNIST dataset, the 10% random perturbation attack realizes a model accuracy of about 81.35%. We hypothesize that the local model accuracy is different from the free-rider accuracy for a normal training lexicon. But the discrepancy is not apparent during the initial training process, and this provides an opportunity for the “free-rider” to maintain its credibility through camouflage, and thus to gain constant access to the global model published by the server.

13.5.2 RQ2: Defensive Effects at Higher Free-Riders Ratios

In the experiments, we observe that over half of the number of clients and over half of the number of free-riders do not have a great impact on the global model’s accuracy under the legacy federation framework. The delta weighting attack and the random perturbation attack can still obtain high-quality models even if the number of free-

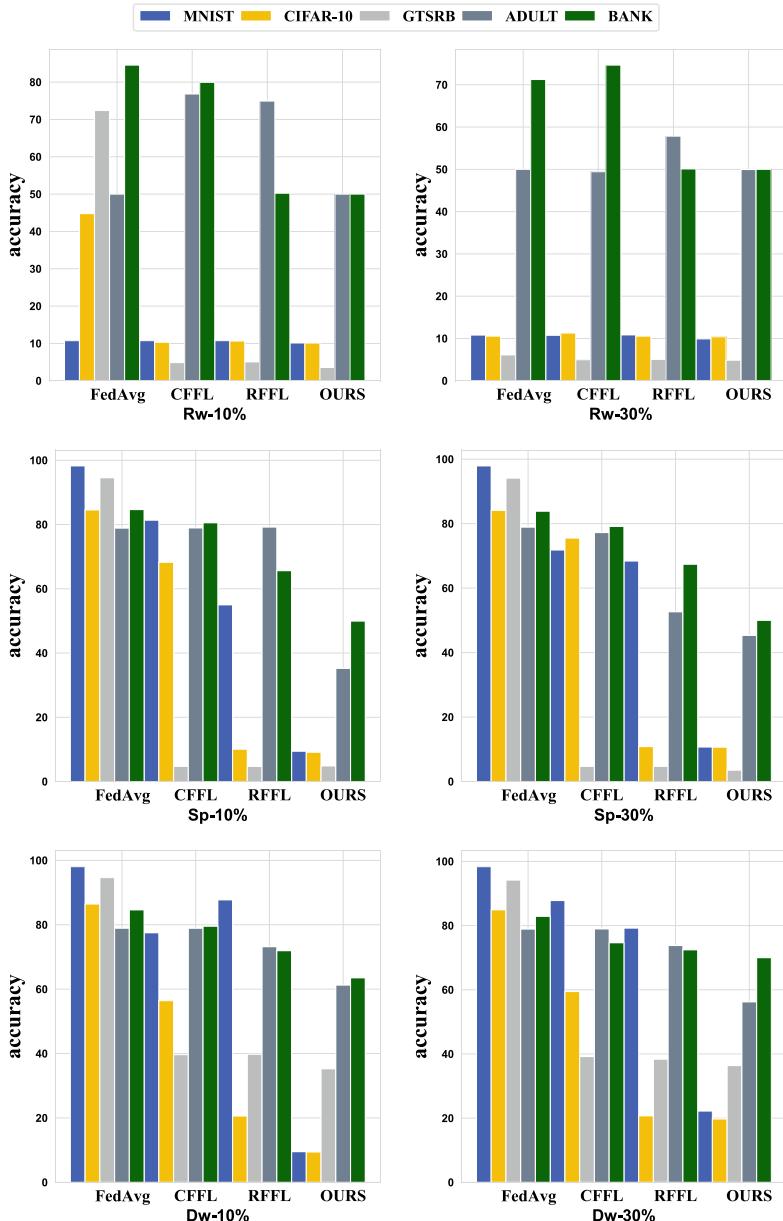


Fig. 13.2 Under the IID data, the comparison between our method and three baselines, with free-riders accounting for 10% and 30% of the clients

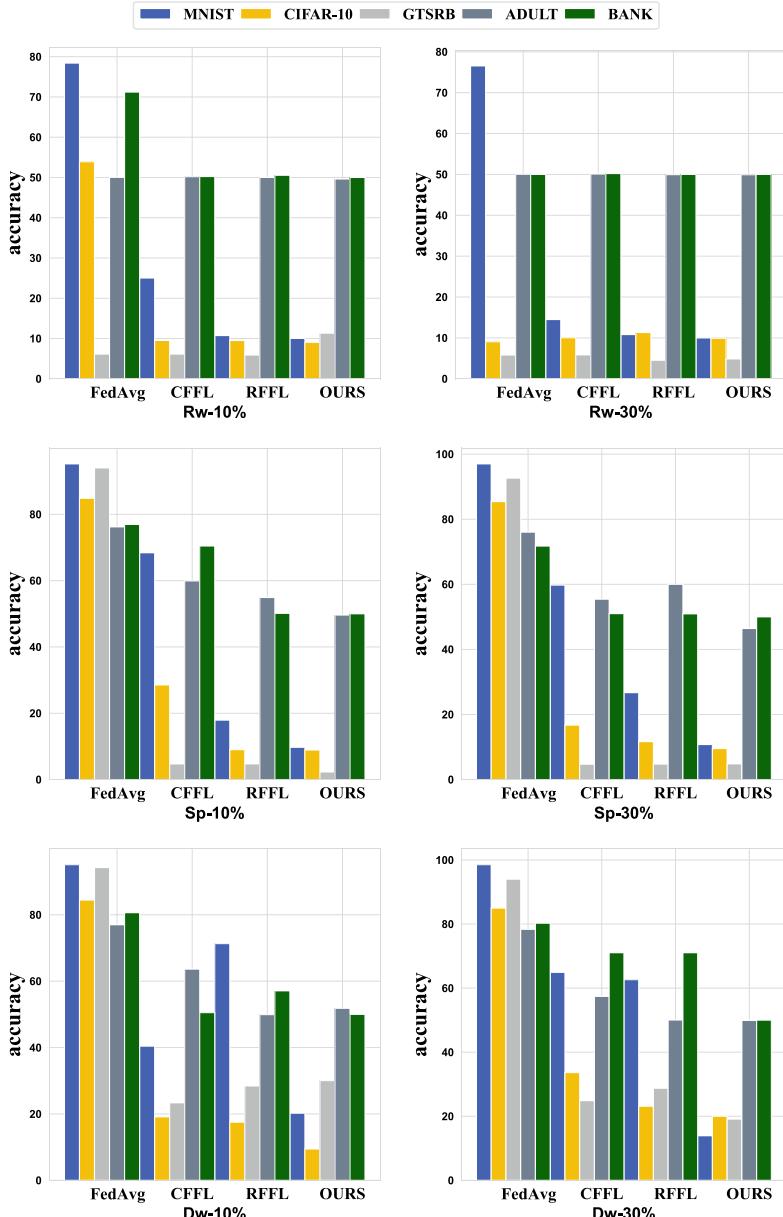


Fig. 13.3 Under the non-IID data, the comparison of our method with the three baselines, free-rider can obtain the highest model accuracy (%). Among them, free-riders account for 10% and 30% of the total number of clients

riders reaches 90% of the total number of customers under the undefended FedAvg aggregation framework, as shown in Figs. 13.4 and 13.5. For instance, free-riders on the MNIST, GTSRB, and BANK datasets in Fig.(13.4) can obtain more than 80% model accuracy overall. This demonstrates that high percentage of free-riders can still be hidden in federated training, thus obtaining a high-quality model. Hence, in this section, however, we shall take into account the possibility that a larger proportion of free-riders may influence the effectiveness of the defense.

Implementation Details. (1) The five datasets were distributed using IID and non-IID distributions. A Dirichlet distribution is used for the non-IID data to explore the issue of dissimilarity, with a distribution coefficient of 0.5 by default. (2) We have set the percentage of free-riders in the 10 clients to 50% and 90%, respectively. And, when the number of free-riders is equal to or much higher than the number of benign clients, this helps to find out how our method performs. Figures 13.4 and 13.5 show the results.

Results and Analysis. The results of the experiments show that the defensive capability of our method still achieves the SOTA defense performance when half or more than half of the clients are free-riders. To be specific, the defense capability of our method makes the HMC obtained by free-riders less than 37% when dealing with the image dataset, the lowest HMC is even only 4.80%, which makes it impossible for free-riders to steal the accurate model. Results of the tests on the two kinds of text datasets make the HMC obtained by free-riders less than 69% on their average, and the lowest HMC is only 45.52%, which fully proves that the defense effect of our method is stable.

As opposed to CFFL and RFFL, the effectiveness of our method's defense remains stable. The reason for this is that our defense approach does not begin with model updates, which can be easily spoofed by "free-riders". As a result, our method looks for differences in the development of client-side model training evolution, effectively avoiding the problem of free-riders disguising model updates. Whereas defensive approaches of CFFL and RFFL basically start from the model changes uploaded by the client, and this is the explanation for their reduced effectiveness against both highly camouflaged free-riders and highly proportional free-riders.

13.5.3 *RQ3: Trade-Off Between Defense and Main Task Performance*

In this section, we discuss whether the defensive approach sacrifices major performance.

Implementation Details. In contrast to normal joint training, our method not only requires each client to upload the weight of the update of the local model but also the matrix of WEFs used to detect free-riders. Hence, we calculate the communication overhead cost of our method to analyze its complexity. The time cost of normal training and our method for training is evaluated and compared in terms of training time cost for five datasets with one training data epoch, as shown in Fig. 13.6.

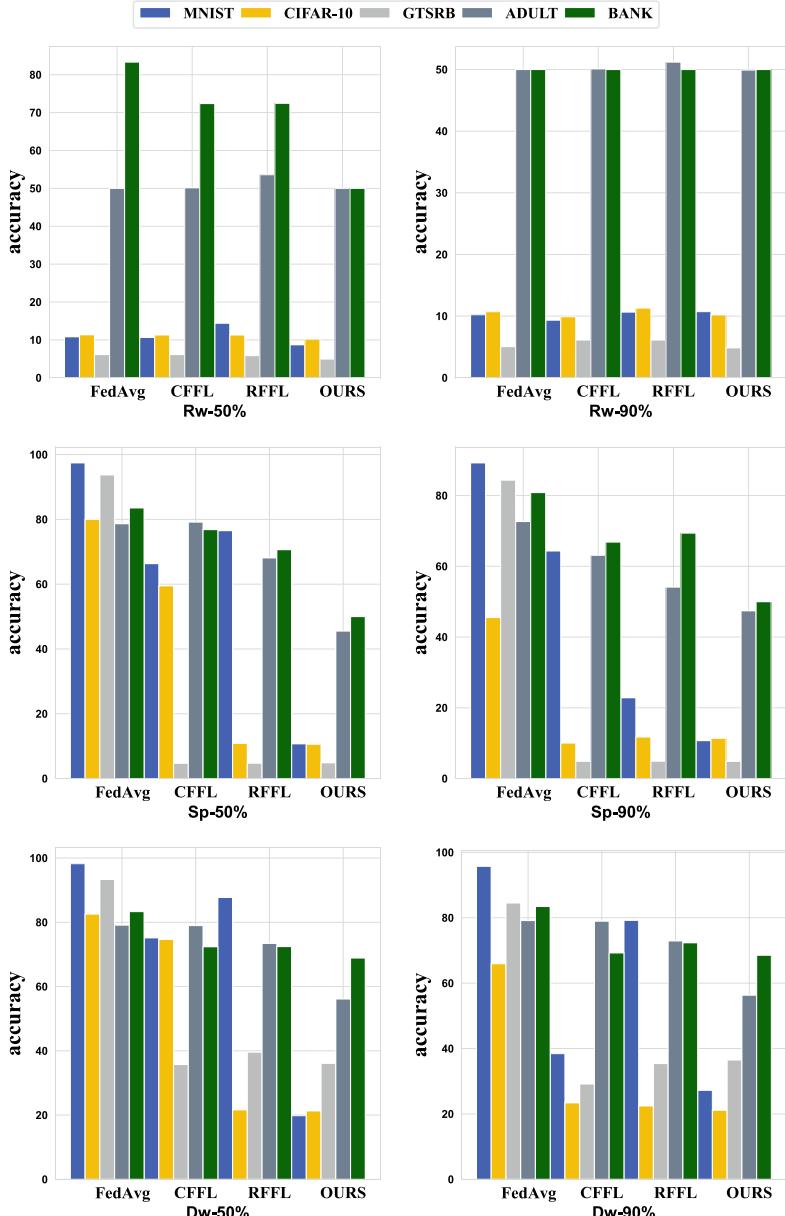


Fig. 13.4 Under the IID data, the comparison of our method with the three baselines, free-rider can obtain the highest model accuracy (%). Among them, free-riders account for 50% and 90% of the total number of clients

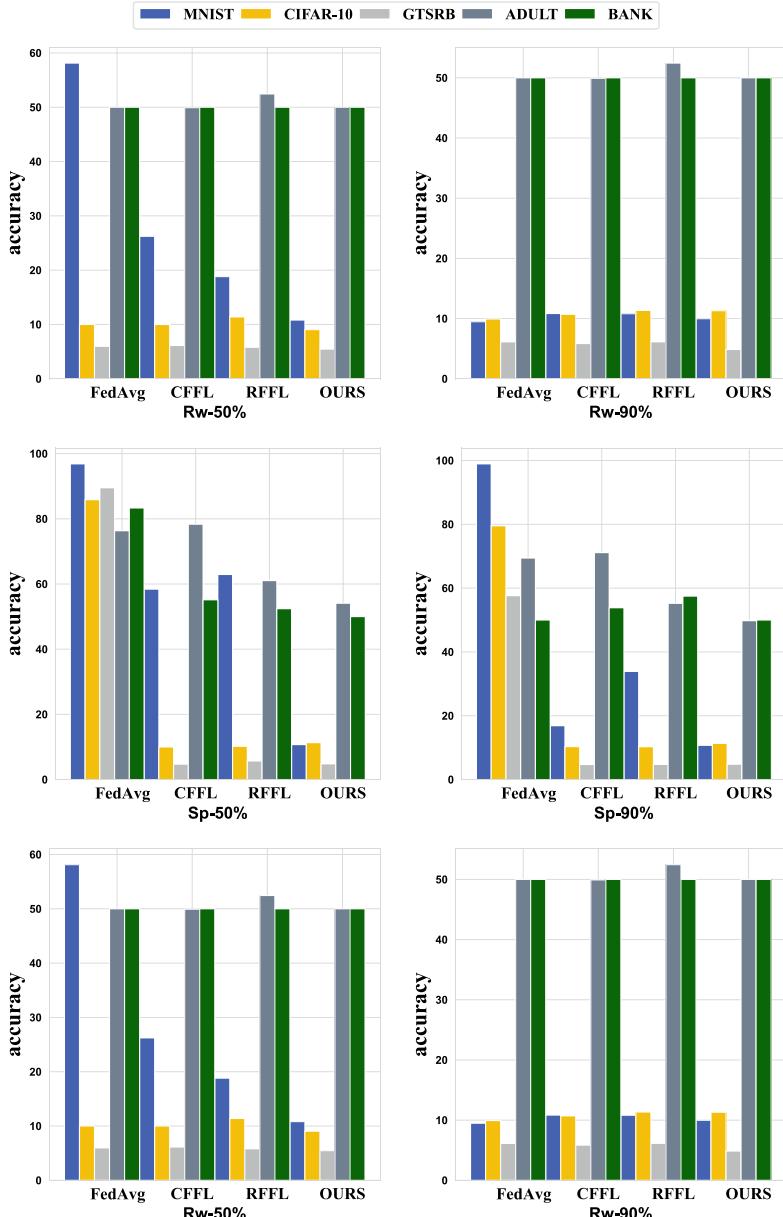


Fig. 13.5 Under the non-IID data, the comparison of our method with the three baselines, free-rider can obtain the highest model accuracy (%). Among them, free-riders account for 50% and 90% of the total number of clients

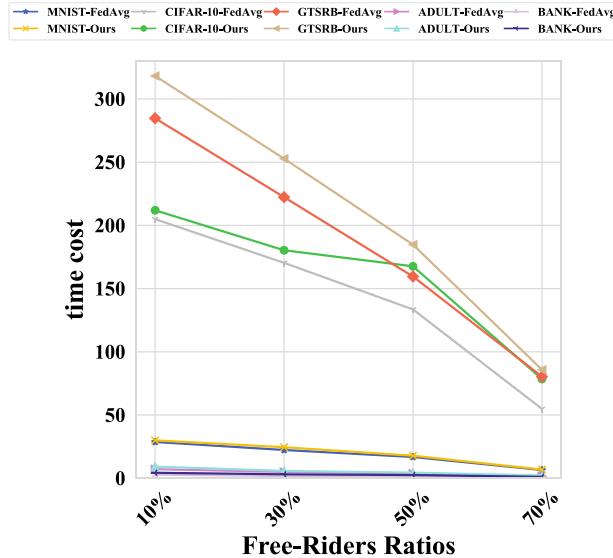


Fig. 13.6 Comparison of our method’s time cost (second) with benign training

Results and Analysis. From Fig. 13.6, the time cost overhead for our method is rather tolerable. For instance, on CIFAR-10 and GTSRB, our method takes 3.41% to 43.15% more time than the normal training process, i.e., Fedavg. On datasets that are easier to train, such as ADULT and BANK, the time cost is basically negligible.

13.6 Conclusion

In this chapter, we emphasize that the variation between free-riders and benign clients in the dynamic training process can be effectively exploited to prevent free-rider attacks, and on this basis, we presented our method. the performance of our method generally outperforms that of all baselines, as well as performs well in defending against various camouflaged “free-rider” attacks. The effectiveness of our method is analyzed further in five aspects, which verify that our method not only defends against hitchhiker attacks but also does not interfere with the training of benign clients. Our method is complementary to existing approaches. In future work, we are planning to explore the potential of combining the two by designing a powerful and more secure federated learning mechanism.

References

1. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20–22 April 2017, Fort Lauderdale, FL, USA. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (2017). <http://proceedings.mlr.press/v54/mcmahan17a.html>
2. Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., Gao, Y.: A survey on federated learning. *Knowl. Based Syst.* **216**, 106775 (2021). <https://doi.org/10.1016/j.knosys.2021.106775>
3. Jiang, C., Xu, C., Zhang, Y.: PFLM: privacy-preserving federated learning with membership proof. *Inf. Sci.* **576**, 288–311 (2021). <https://doi.org/10.1016/j.ins.2021.05.077>
4. Wang, F., Zhu, H., Lu, R., Zheng, Y., Li, H.: A privacy-preserving and non-interactive federated learning scheme for regression training with gradient descent. *Inf. Sci.* **552**, 183–200 (2021). <https://doi.org/10.1016/j.ins.2020.12.007>
5. Jiang, J., Cui, B., Zhang, C.: Distributed Machine Learning and Gradient Optimization. Springer, Berlin (2022). <https://doi.org/10.1007/978-981-16-3420-8>
6. Chen, H.: Reliable and efficient distributed machine learning. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden (2022). <https://nbn-resolving.org/urn:nbn:se:kth:diva-310374>
7. Long, G., Tan, Y., Jiang, J., Zhang, C.: Federated learning for open banking. In: Yang, Q., Fan, L., Yu, H. (eds.) Federated Learning - Privacy and Incentive, Lecture Notes in Computer Science, vol. 12500, pp. 240–254. Springer (2020). https://doi.org/10.1007/978-3-030-63076-8_17
8. Shengi, G.: A federated learning based approach for loan defaults prediction. In: Fatta, G.D., Sheng, V.S., Cuzzocrea, A., Zaniolo, C., Wu, X. (eds.) 20th International Conference on Data Mining Workshops, ICDM Workshops 2020, Sorrento, Italy, November 17–20, 2020, pp. 362–368. IEEE (2020). <https://doi.org/10.1109/ICDMW51313.2020.00057>
9. Xu, J., Glicksberg, B.S., Su, C., Walker, P.B., Bian, J., Wang, F.: Federated learning for healthcare informatics. *J. Heal. Inf. Res.* **5**(1), 1–19 (2021)
10. Kuo, T., Pham, A.: Detecting model misconducts in decentralized healthcare federated learning. *Int. J. Med. Inf.* **158**(February), 104658 (2022). <https://doi.org/10.1016/j.ijmedinf.2021.104658>
11. Hard, A., Rao, K., Mathews, R., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D.: Federated learning for mobile keyboard prediction. *CoRR* **abs/1811.03604** (2018). <http://arxiv.org/abs/1811.03604>
12. Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D., Beaufays, F.: Applied federated learning: improving google keyboard query suggestions. *CoRR* **abs/1812.02903** (2018). <http://arxiv.org/abs/1812.02903>
13. Lin, J., Du, M., Liu, J.: Free-riders in federated learning: attacks and defenses. *CoRR* **abs/1911.12560** (2019). <http://arxiv.org/abs/1911.12560>
14. Zhao, Z., Huang, J., Roos, S., Chen, L.Y.: Attacks and defenses for free-riders in multi-discriminator GAN. *CoRR* **abs/2201.09967** (2022). <https://arxiv.org/abs/2201.09967>
15. Huang, W., Li, T., Wang, D., Du, S., Zhang, J., Huang, T.: Fairness and accuracy in horizontal federated learning. *Inf. Sci.* **589**, 170–185 (2022). <https://doi.org/10.1016/j.ins.2021.12.102>
16. Gao, L., Li, L., Chen, Y., Xu, C., Xu, M.: FGFL: A blockchain-based fair incentive governor for federated learning. *J. Parallel Distrib. Comput.* **163**, 283–299 (2022). <https://doi.org/10.1016/j.jpdc.2022.01.019>
17. Fraboni, Y., Vidal, R., Lorenzi, M.: Free-rider attacks on model aggregation in federated learning. In: The 24th International Conference on Artificial Intelligence and Statistics, AISTATS 2021, April 13–15, 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 130, pp. 1846–1854. PMLR (2021). <http://proceedings.mlr.press/v130/fraboni21a.html>

18. Zong, B., Song, Q., Min, M.R., Cheng, W., Lumezanu, C., Cho, D., Chen, H.: Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, Conference Track Proceedings. OpenReview.net (2018). <https://openreview.net/forum?id=BJJLHbb0->
19. Lyu, L., Xu, X., Wang, Q., Yu, H.: Collaborative fairness in federated learning. In: Federated Learning - Privacy and Incentive, Lecture Notes in Computer Science, vol. 12500, pp. 189–204. Springer (2020). https://doi.org/10.1007/978-3-030-63076-8_14
20. Sermanet, P., LeCun, Y.: Traffic sign recognition with multi-scale convolutional networks. In: The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California, USA, July 31–August 5, 2011, pp. 2809–2813. IEEE (2011). <https://doi.org/10.1109/IJCNN.2011.6033589>
21. Kohavi, R.: Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, pp. 202–207. AAAI Press (1996). <http://www.aaai.org/Library/KDD/1996/kdd96-033.php>
22. El-Sawy, A., El-Bakry, H.M., Loey, M.: CNN for handwritten arabic digits recognition based on lenet-5. In: Proceedings of the International Conference on Advanced Intelligent Systems and Informatics, AISI 2016, Cairo, Egypt, October 24–26, 2016. Advances in Intelligent Systems and Computing, vol. 533, pp. 566–575 (2016). https://doi.org/10.1007/978-3-319-48308-5_54
23. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, Conference Track Proceedings (2015). <http://arxiv.org/abs/1409.1556>
24. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 770–778. IEEE Computer Society (2016). <https://doi.org/10.1109/CVPR.2016.90>
25. Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M., Dosovitskiy, A.: Mlp-mixer: an all-mlp architecture for vision. CoRR **abs/2105.01601** (2021). <https://arxiv.org/abs/2105.01601>
26. Xu, X., Lyu, L.: A reputation mechanism is all you need: collaborative fairness and adversarial robustness in federated learning. In: Proceedings of the ICML Workshop on Federated Learning for User Privacy and Data Confidentiality (2021)

Chapter 14

An Effective Model Copyright Protection for Federated Learning



14.1 Introduction

Federated learning [1–3] is a framework that uses distributed learning to protect user data privacy. Because of its privacy and high efficiency, FL has been widely applied to practical scenarios such as bank loans [4], medical diagnosis [5], and recommendation systems [6]. Unfortunately, there is no appropriate way to protect the copyright of FL-trained models. This problem makes it difficult for the owner of the model to claim the copyright even though it is suspected that the model is being used for profit. How to protect the ownership of the model and the benefits of the parties is a pressing problem.

There are several approaches to model copyright protection in a centralized scenario. Watermark embedding [7–9] and model fingerprinting [10, 11] are the two mainstream model ownership verifications. They require the model owner to embed a unique secret watermark or create a unique secret fingerprint before releasing the model. In the case that the model owner wants to claim ownership of the model, the owner can make use of the signature or fingerprint to prove it. However, it is necessary to modify the model parameters when embedding a watermark, which will inevitably influence the performance and efficiency of federal training. Model fingerprints methods extract model features (e.g., gradients) to generate an adversarial example [12, 13] and use the transferability of the generated adversarial examples to determine whether the use of the suspect model is unauthorized. However, fingerprints are easily erased by adversarial retraining [11].

Different from centralized training, FL is distributed and involved in server and client. According to the FL's assumption, however, the client may not be trusted but the server is, that is, a trusted and honest server in FL. Therefore, it is more reasonable to depend on the server for copyright protection rather than for the client. However, the direct migration of centralized copyright protection to FL is still a challenge. We summarize the main challenges addressed for the model copyright protection in FL: (i) *Data limitation*—Honest servers are unable to access the client's own training data to create watermarks or model fingerprints. (ii) *Accuracy sacrifice*—The pro-

tention should not unduly sacrifice the accuracy of the server model and the training efficiency of the FL. (iii) *Malicious attacks*—The server-side protection approach is designed to handle downstream attacks, such as fine-tuning, model pruning, and malicious clients upstream in collaborative training. (iv) *Black-box ownership verification*—For black-box ownership verification scenarios (that is, with no knowledge of the structure or parameters of the model), the model copyright verification should be practical, e.g., only the Application Programming Interface (API) calls are available to query the suspicious model.

To address these challenges, we propose the first approach to use model fingerprints to protect model copyright in FL. Our method is based on model fingerprint techniques but with improved robustness against adversarial retraining attacks. Generally, we rely on the honest FL server to generate fingerprints for the FL-trained model and train a separate model using these fingerprints and their outputs. Specifically, by using secret key samples, the server leverages on the extracted global model features and generates a set of adversarial examples as model fingerprints. Note that the key samples here are not necessarily of the same distribution with the training or testing samples. Then we use these fingerprints as input to the model and obtain the feature distributions of these fingerprints. The feature distribution of the key samples is used to train a new model, i.e., the detector, which is to predict the ground-truth class encode of the key samples. When testing a suspect model, we obtain the feature distribution of the key samples output by the suspect model and use the detector to predict the key samples' ground-truth class. When the accuracy of the detector exceeds a threshold, the ownership of the model is claimed.

Moreover, our method has designed the model fingerprint to have adaptive enhancement capability. It gradually adds model features in response to the changes in the global model during federation training. During the verification phase, our method only accesses the model's output of key samples, which is suitable for the black-box forensic scenarios. We conducted extensive experiments to evaluate the validity, fidelity, efficiency, and robustness of our method.

The main contributions of this chapter are summarized as follows.

- We propose the FL-oriented model copyright protection method by relying on the honest server to generate robust fingerprints without any knowledge of the training data on the client side.
- Our method innovatively introduces a detector to capture the relationship between the feature distribution of the key samples output by the target model and the key samples' label. Our method can effectively verify the ownership of the target model according to the accuracy of the detector, i.e., measuring to what extent a suspect model's behavior on these key samples aligns exactly with the target model.
- We conducted extensive experiments to evaluate our method on three datasets and nine models. The experimental results show the advantages of our method compared to previous work and satisfy validity, fidelity, robustness, and black-box ownership verification in FL scenarios.

14.2 Related Works

14.2.1 Centralized Model IP Protection

The current IP protection methods are mainly used in deep neural network (DNN). IP protection methods are divided into two streams: watermark and fingerprint. Since Uchida et al. [7] first proposed a watermarking model framework in a white-box scenario. It is further improved by limiting access rights in a black-box scenario. Li et al. [14] combine common data samples with exclusive “logos” and train models to predict them as specific labels so that a third party can verify the ownership of the model. To protect against copyright fraud, Li et al. [15] proposed a blind watermark algorithm to generate key samples that are similar to the original samples. However, the embedding of watermarks changes the model parameters, which will inevitably influence the performance of the model. Fingerprint is another type of IP protection, which does not alter model parameters. Zhao [10], Lukas, et al. [11] as model fingerprints. Fingerprints are generated by the extraction of model features, so they do not affect the performance of the model. However, it is possible to remove model fingerprints in the face of adaptive adversarial training. Furthermore, the privacy of FL, which does not have any training data, poses a huge challenge to the process of fingerprinting on the server.

14.2.2 IP Protection in FL

The mainstream IP protection in FL is still based on watermarking [16–18]. Specifically, Tekgul et al. [16] proposed WAFFLE which achieves IP protection by embedding watermarks on the server. Bowen Li et al. [17] proposed FedIPR which embeds and detects watermarks by each client independently. Fang et al. [18] proposed the Merkle-Sign watermark scheme, which is a combination of the most advanced watermarking scheme and a secure scheme for distributed storage. However, they either suffer the inherent limitation of the watermarking scheme or pose changes to the training of federated learning, both of which lead to the negative impact on the model performance.

14.3 Preliminaries and Background

In this section, we introduce the horizontal federation framework and the background knowledge for generating model fingerprints.

14.3.1 Horizontal Federated Learning

Horizontal federated learning [19] can be used to solve problems in which the data sets of each client have the same feature space and different sample spaces. All personal data is on the client, and no other client can access it. The model parameter w_i is uploaded to the server after each client C_i performs the model training locally. Then, the server aggregates the parameters that have been uploaded to form a global model parameter w_g , which is then sent back to each client for further training. The commonly used aggregation rule is as follows:

$$w_g = \frac{1}{K} \sum_{i=1}^K w_i \quad (14.1)$$

where K is the total number of clients participating in training.

In HFL, only information about locally trained model is shared. It thus ensures the privacy of the client's local data.

14.3.2 Key Samples

Key samples are the seeds to produce model fingerprints. Key patterns are usually protected against intrusion. Existing model protection methods are based on the key samples from the training data. This may be a problem with FL, however, as the server in FL has no access to client data, so the training data cannot be used as a key sample. If the key samples are part of the training data, it is very easy for the attackers to use the adversarial retraining method to remove the fingerprints. Thus, the key samples are not related to the training data. This could 1) fits in the FL scenario when the training data are not available; and 2) significantly affects the performance of the main task if the attacker performs the retraining attack.

It is also possible to generate effective model fingerprints by using non-training data as key samples, since the fingerprint of our method is made up of key samples and model-specific characteristics. In particular, the key sample is only a carrier of the model fingerprint, which is dependent on the feature of the model. Thus, we propose an efficient method to generate fingerprints without training data.

14.3.3 Model Fingerprints

Model fingerprinting is a method to generate a model fingerprint F based on the target model.

It includes the following two algorithms:

Generate model fingerprints. $F = \text{Generate}(G; D_{key})$. The generation process has access to global model G and key samples D_{key} , and uses this knowledge to generate the model fingerprint F for a specific label F_y .

Validate model fingerprints. $F_y^{pre} = Validate(G_{sus}; F)$. The model owner validates the suspect model G_{sus} using the model fingerprint F , and the obtained output prediction label F_y^{pre} are compared with F_y to verify the ownership.

The evaluation of the suspect model ownership using the validation algorithm is based on an empirically determined threshold α . If the matching rate between F_y^{pre} and F_y is greater than α , it is verified as a stolen model. Since the model fingerprint, i.e., the adversarial example, has transferability, the model ownership can still be verified even though the suspect model is slightly modified from the original model.

Existing model fingerprint verification algorithms only using prediction labels are vulnerable to ambiguity attacks, where a fake owner uses samples with the same output label as the original model fingerprints to falsely claim the model ownership. For example, the model owner is considered to have ownership of the model using a key sample output label “1”. However, the attacker uses a non-key sample output label also “1”, which is an eventuality and makes the ownership of the model confusing. We take this into account when designing our fingerprint verification algorithm. We use the output distribution features of the model instead of prediction label only, which will be detailed in Sect. 14.5.4.

14.4 Threat Model

This chapter takes the server as the initiator of property protection, and the main threat target is the upstream and downstream attackers.

Suppose the horizontal federation contains K clients C_i and a trusted server, where each client has local data D_i , $i \in \{1, 2, \dots, K\}$. All clients collaborate to train a global model G , but the distributed mechanism brings uncertainties, such as the existence of malicious client C_j^m during the training process who intends to exploit the high-value global model G for illegal profit. Moreover, when the federation training is finished, the public sharing of the global model leads to a downstream attacker P_m being able to tune it to avoid the owner’s IP rights traceability and then uses the tuned model for other illegal avenues. To clearly describe such a threat scenario, we give the formal definitions of attacker and defender as follows.

Attacker ability. We suppose the attacker can be the upstream malicious client C_j^m or the downstream malicious party P_m . The upstream malicious client C_j^m has the generally accessible information set $\langle \Theta_g, L, l_r, R_{aggr} \rangle$ in the FL system, where Θ_g is the global model parameter, L is the loss function, l_r is the learning rate and R_{aggr} is the aggregation rule. However, C_j^m cannot access or manipulate other clients’ data D_i . The downstream malicious party P_m can generally only have the global model parameter Θ_g information. But both attackers can use some modification attacks against the global model (e.g., model fine-tuning [7], model pruning [20]), denoted as $\Theta'_g = Mod(\Theta_g)$, thereby making it different from the original model $G(\Theta_g) \neq G(\Theta'_g)$, to prevent the trace verification of model IP protection, but still has high accuracy:

$$\|Acc(D_{test}; G(\Theta'_g)) - Acc(D_{test}; G(\Theta_g))\| < \delta \quad (14.2)$$

where δ is a small number.

Defender ability. The goal of the defender P_d is to protect the IP of models collaboratively trained by multiple clients in the FL scenario. In a practical scenario, the P_d has no white-box access to the model stolen by the attacker. However, the P_d should have black-box access to the suspect model G_{sus} , i.e., it can query the suspicious model G_{sus} with the $< X_{key}, Y_{key} >$ and obtain the output $Y_{pre} \leftarrow f(X_{key}, G_{sus})$ to verify the model's IP $Ver(< Y_{pre} = Y_{key} ? >; G_{sus})$. A rational assumption would be 1) the defender has the white-box access to the model to be protected, since he/she owns the copyright; 2) the defender only has black-box access to the suspect model deployed by others.

14.5 Methodology

We rely on the server to generate fingerprints for the purpose the model copyright protection. An overview of our method is shown in Fig. 14.1. The overall process consists of two phases, i.e., fingerprint generation and copyright verification. Specifically, the first phase is divided into three steps: ① generate adversarial examples as model fingerprints; ② obtain the output distribution features of the model fingerprints and label them with specific labels; and ③ detector training. The validation stage takes place when there is a suspicious model and the owner of the model wants to verify the ownership of the model. It is necessary for the owner to get the characteristic distribution of fingerprint samples through the query of the suspect model. Then, we feed the results to the detector to see if the prediction precision is high enough to claim ownership of the model.

14.5.1 Model Fingerprints Generation

The server has no access to the local data of the client, but can know the concrete training task clearly. Using this knowledge, we have collected a number of key samples, D_{key} , which have nothing to do with training data. For example, if the server is aware that the primary task of the federation training is Handwriting Digit Recognition, it can take the face image as the key sample. Class number of key samples D_{key} is also chosen according to the number of classes for the main task. In general, it is preferable to increase the number of key sample classes in order to improve performance, for example, with a high detection rate and a low false positive rate.

Our method has no dependency on the key samples. There are two major parts to generate the model fingerprint, one is the selected key sample and the other is the feature extraction of the model. The key sample is just a carrier, but the key point

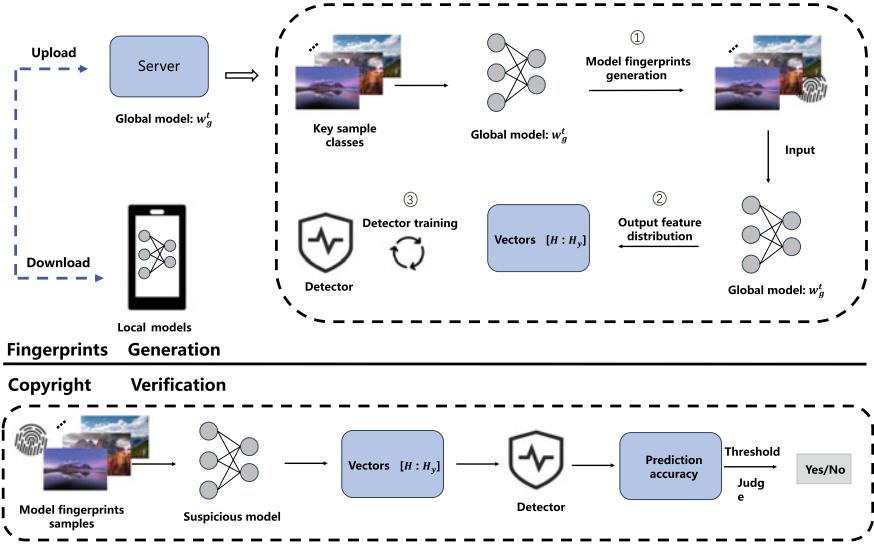


Fig. 14.1 Overview of our method. The server clones a global model, extracts the inherent features of the model, and generates adversarial examples from key samples as model fingerprints. After input to the global model, the output distribution features are obtained and labeled to train the detector. Using the transferability of the adversarial example, model owners can use the detector to determine the ownership of the suspect model

is to extract the feature of the model, which reflects the specific information of the model. In general, it is necessary that the key samples are not related to the federation training task. This is to prevent the attacker from performing adversarial retraining attacks.

The selected key pattern D_{key} is used to create model fingerprints F . Specifically, using adversarial attacks, we combine key samples D_{key} and extracted model perturbations to form adversarial examples D_{adv} as model fingerprints. The objective of the adversarial attack is to minimize the interference of normal examples and to maximize the likelihood of misleading the classifier. This can be modeled as an optimization problem. In this paper, we present a general DNN model called ρ -loss, defined as

$$\begin{aligned} & \operatorname{argmin} \{ \epsilon \|\rho\| + \lambda Loss(y, f_{pre}(\Theta, x)) \} \\ s.t. \quad & \rho = X_{adv} - X_{key} , \quad X_{adv} \in \{D_{adv}\} , \quad X_{key} \in \{D_{key}\} , \quad y \in \{y_{orig}, y_{target}\} \end{aligned} \quad (14.3)$$

where ϵ is the scale factor used to balance the order-of-magnitude difference in the perturbation; ρ denotes the perturbation between the adversarial example X_{adv} and the key example X_{key} . X_{adv} is the sample in the set of adversarial examples D_{adv} , and

X_{key} is the sample in the set of key examples D_{key} ; $Loss(\cdot, \cdot)$ is the loss function of the model; $f_{pre}(\cdot, \cdot)$ is the prediction result of the model; Θ denotes the parameters of the model; x and y denote the input key example D_{key} and the corresponding output class labels, respectively. When $\lambda=1$ and $y = y_{target}$, ρ -loss represents a targeted attack, and when $\lambda=-1$ and $y = y_{orig}$, ρ -loss represents an untargeted attack.

The targeted attack has a clear optimization direction and better attack effect compared with the untargeted attack. Thus, we use targeted attacks by setting $\lambda=1$, and generate model fingerprints F for the corresponding classes. It is worth noting that our method is a generic framework transferable for other adversarial attacks including C&W [21], PGD [22], etc.

The Mean Square Error (MSE) in the oscillation period T increases with the training epoch, but there is a time delay t_d on the growth curve. The MSE of the model fingerprint is basically constant during linear growth. During the oscillation period T , we conclude that the model is rapidly learning the “knowledge” in the data. As a result, the model fingerprint is able to quickly gather the features trained by the model, and this is reflected in the increase of the MSE. After the model learns the “knowledge”, the feature of the model fingerprint is collected, so there is a time delay of t_d . In the course of linear growth, the model only keeps the distance between classes and increases the distance between classes. We conclude that at this point the model fingerprint can be provided with fewer features, so the MSE value of the model fingerprint remains substantially unchanged. In summary, our proposed model fingerprint has a direct relationship with the global model, and the global model is unique.

Regarding the timing of model fingerprint generation, some options are possible, such as generating fingerprints during the training process or after the training. If the generation of model fingerprints starts at the end of federated training, since the client gets the global model published by the server in each epoch of training, then it is possible for the malicious client to save the global models of previous epochs locally to evade IP verification. Therefore, the generation of model fingerprints needs to be performed during the federated training. The other way is considering the model fingerprint generated from the beginning to the end of the whole federated training. It still brings some problems, not only to consider the time overhead of IP protection but also at the early stage of training, the parameters of the global model change sharply with the iterative update in FL, thus the model fingerprint generated in the previous epoch to be input to the global model in the next epoch of training will lead the target label shifted, resulting in great change in the output distribution vector, i.e., the model fingerprint is invalid. As shown in Fig. 14.2a, we randomly select the model fingerprint with target label “5” generated in epoch 50 and show the visualization of its output distribution vector in epochs 50–52. It can be found that the model fingerprint generated in epoch 50 has been invalidated in the subsequent epochs. It indicates that the static generation method is difficult to maintain the validity of fingerprints in dynamic training.

To solve this problem and ensure the stability of the model fingerprint, we propose a model fingerprint adaptive enhancement mechanism.

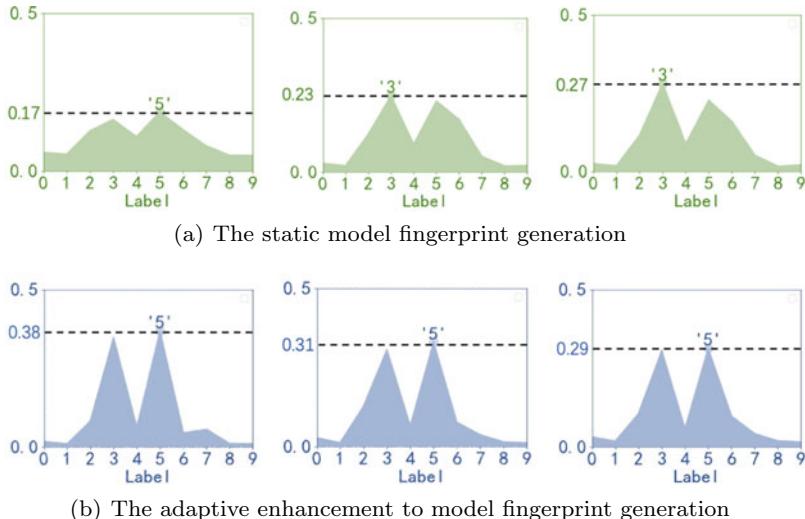


Fig. 14.2 The output distribution feature vectors are visualized for a LeNet-5 model trained on the MNIST dataset. The fingerprints are generated in the 50th epoch and verified in the 50th, 51st, and 52nd epochs

14.5.2 Model Fingerprinting Adaptive Enhancement

The model parameters usually change sharply at the early stage of FL training, and meanwhile, the initial model accuracy is low and not of high value for use. Therefore, fingerprint generation should be executed from the middle stage of training. The middle stage of performing fingerprint generation is not absolute, but rather a relative range interval. For the model parameter update rate, it can be found that the global model parameter change rate increases rapidly in the first 5 epochs, decays rapidly from epochs 5–20, and gradually levels off afterwards. For the threshold α , the α starts to gradually decrease in epochs 0–30 as the timing of selecting the execution of generating fingerprints shifts backward; after 30 epochs, the α starts to increase. This indicates that in the preliminary model training, the initialized model parameters take some time to learn the features of the data, and the parameters are highly variable and not stable enough in this process, resulting in high α . As the timing of generating fingerprints moved backward, the α began to gradually decrease. It is not until after 30 epochs that the α starts to increase, which may be due to the fact that as the overall number of epochs to generate fingerprints decreases, the amount of data generated is decreasing, which can affect the training of the detector M . It is worth noting that the overall α is not higher than 28%, which is within the acceptable range. To sum up, the fingerprint generation is performed starting in the middle of training having a lower threshold α .

On the other hand, adaptive enhancement of fingerprints is achieved by setting a fingerprint validity detection checkpoint, which is defined as follows:

$$F^{t+1} = \begin{cases} D_{adv}^t + \epsilon \|\rho\|, & \text{Checkpoint}(D_{adv}^t) \times \\ D_{adv}^t, & \text{Checkpoint}(D_{adv}^t) \checkmark \end{cases} \quad (14.4)$$

where $\text{Checkpoint}(\cdot)$ is to determine whether the target label of the adversarial sample has been shifted, i.e., whether the fingerprint is invalidated; $\epsilon \|\rho\|$ denotes adding the feature perturbation of the current epoch model; The “ \times ” indicates invalid, and the “ \checkmark ” indicates valid.

In each epoch, we use Algorithm 14.1 to judge whether the previous model fingerprint is valid and if it is invalid, we continue to add the model feature perturbation in the current epoch to achieve the effect of dynamic enhancement.

From Fig. 14.2b, we can also see that the model fingerprint generation algorithm with adaptive enhancement can maintain better across epochs, which effectively improves the validity of the generated fingerprints.

14.5.3 Detector Training

The detector M uses the MLP structure, whose training data is constructed from the vector H of feature distributions obtained from the model fingerprint F input to the global model G . And H is labeled with a specific label H_y to train the detector M , where H_y is consistent with the model fingerprint label, i.e., H_y corresponds to the label of D_{adv} .

$$H = G(X_{adv}) , \quad X_{adv} \in \{D_{adv}\} \quad (14.5)$$

Our method is robust to model modification attacks. Since the detector M does not depend directly on the parameters of the model, but instead trains the detector according to the output characteristic H of the model fingerprint, the output characteristic H will vary with the model, so that the detector M can be dynamically trained. Thus, it realizes a property right statement for the model. When the model is modified, there is no significant change in the output characteristics of the model, so it can be used to validate the model property.

Training the detector M with the output distribution feature H effectively eliminates the risk of a malicious client using an ambiguous attack in the property rights verification process, that is, using a non-key sample that is identical to a key sample's output tag for the purpose of obfuscation. This is because there is no correlation between the behavior of the key sample H and the non-key sample.

In order to improve the performance of the detector M , we can train it by increasing the number of key sample classes c to generate a variety of adversarial examples.

$$\text{Train}(< H, H_y >; M(w_m, c)) \quad (14.6)$$

where $Train(., .)$ represents the training of the detector M using the feature distribution vector H and the given label H_y . w_m denotes the detector model parameters and c denotes the number of classes of key samples.

The training of the M detector is independent of the main training task of FL. The current global model is cloned to generate the fingerprint of the F model and train the M detector. This independence means that copyright protection does not sacrifice model accuracy.

14.5.4 IP Verification

The trained detector M can be used to identify the attributes of the suspicious model. In the validation phase, the fingerprint of the model is fed into the suspicious model and the output distribution vector H is fed into the detector M to check whether the output label matches the set label H_y . The model is characterized based on whether the accuracy of the detector is above a set threshold α . In this process, only the suspect inputs and outputs of the model are used without accessing the internal parameters of the model. Algorithm 14.1 describes the details of this approach.

The effectiveness of property verification stems from the development of model fingerprints, which reflect model-specific “fingerprint information”. A trained detector effectively captures the specific output characteristics of the model fingerprint.

Algorithm 14.1: Our method.

Input: total number of clients K ; dataset $\{D_i\}$ of each client, $i \in \{1, 2, \dots, K\}$; key samples $\{D_{key}\}$ in the server; the global epochs t ; initial global model parameters $w_g^{t=0}$, detector parameters $w_m^{t=0}$; hyperparameter $\epsilon = 0.05$.

Output: the global model G and detector M .

1. Initialization: local model $w_i^{t=0} = w_g^{t=0}$.
 2. **Role:** Client C_i
 3. $w_i^{t+1} \leftarrow Update(w_g^t)$
 4. Local updates w_i^{t+1} upload to Server
 5. **Role:** Server
 6. Calculate w_g^{t+1} according to Eq. (14.1)
 7. **Fingerprints generation:**
 8. Generating F according to Eq. (14.3)
 9. Get the output vector H of F according to Eq. (14.5)
 10. Enhancing F according to Eq. (14.4)
 11. Train the M according to Eq. (14.6)
 12. **Return:** the global model G and detector M
-

14.5.5 Algorithm Complexity

We analyze the complexity of our method on client side and server side, respectively. On the client side, the time complexity is mainly dependent on the number of training epochs for the local model.

$$T_{client} \sim \mathcal{O}(t_l) \quad (14.7)$$

where t_l is the local training epochs.

On the server side, the server implements two main working parts that are independent of each other. The first part is mainly responsible for summarizing the model parameters uploaded by clients and distributing them to each client.

$$Part_1 : T_{server} \sim \mathcal{O}(K) \quad (14.8)$$

where K is the number of clients participating in the training.

The second part of the work focuses on the IP declaration process, i.e., the time complexity of our framework. It consists of three parts: (1) generating the model fingerprints F ; (2) detecting whether the model fingerprints are invalid and performing enhancements; (3) training the detector M .

$$Part_2 : T_{server} \sim \mathcal{O}(Iter * n) + \mathcal{O}(n + Iter * n_{inv}) + \mathcal{O}(t_m) \quad (14.9)$$

where $Iter$ is the number of optimization iterations, n is the number of key samples, n_{inv} is the number of invalid model fingerprints and t_m is the number of training epochs of the detector M .

The time complexity of the framework is low. The only overhead introduced by this approach is $Part_2 : T_{server}$. The choice of output characteristics such as samples used for detector training and their parameter sizes is acceptable and the detector training complexity is low. In addition, the feature declaration process of the method does not affect collaborative learning and each other, effectively mitigating the impact on the efficiency of collaborative learning.

14.6 Experiments Design and Setup

Datasets: We evaluate our method on three datasets, i.e., MNIST [23], CIFAR-10 [24] and CIFAR-100 [25]. MNIST dataset contains 70,000 real-world handwritten images with digits ranging from 0 to 9. Both the CIFAR-10 and CIFAR-100 datasets contain 60,000 color images of size 32×32 , with 10 classes of 6,000 images each for CIFAR-10 and 100 classes of 600 images each for CIFAR-100.

Number of clients: We adopt five clients for FL training in all experiments except the parameter sensitivity experiments with different client numbers.

Models: A number of classifiers are used for verification of various datasets. For MNIST, LeNet-1, LeNet-4, and LeNet-5 [26] are used for classification. For more complex image datasets, CIFAR-10 and CIFAR-100, VGG-11, VGG-13, VGG-16 [27], and ResNet-18 [28], ResNet-34, ResNet-50 are adopted, respectively. All evaluation results are the average of ten runs under the same setting.

Adversarial Attacks: In the main experiments, a targeted attack method based on FGSM is used. Meanwhile, in sensitivity analysis experiments, C&W and PGD adversarial attack methods are adopted to illustrate the transferability of our method.

Hyperparameters: For all experiments, we set the hyperparameter $\epsilon = 0.08$, and select the key sample of $c = 10$.

Attack Methods: In order to evaluate the robustness of the method, we use two well-known model modification attacks, i.e., fine-tuning and model pruning, and an adaptive attack, i.e., adversarial retraining. In addition, malicious client attacks during training are considered and copyright circumvention attacks are analyzed.

FtuningAtt (Model Fine-tuning Attack [7]): It is a classic approach, allowing an attacker to re-train the model with comparable performance to the original model, but of different parameters. In this chapter, the model is fine-tuned by using 10% of the data in the test set.

PruningAtt (Model Pruning Attack [20]): It is designed to cut down targeted parameters and to obtain a new model that is different from the original model but still has similar accuracy. We use the pruning algorithm in [20], setting a certain percentage of the parameters with the smallest absolute value to 0. The percentage is set between 30% and 90% with an interval of 10%.

AdaptiveAtt (Adaptive Attack): We consider the AdaptiveAtt to eliminate the fingerprints and perform adversarial retraining of the model according to the different knowledge possessed by the attacker.

CollabAtt (Collaborative Attack): Malicious clients can collaborate, including multiple upstream clients collaborating and upstream and downstream malicious parties to combine to damage IP.

MaliClientAtt (Malicious Client Attack): In comparison with the WAFFLE method, the malicious client is set up to perform removal attack [29] during the fingerprint generation phase to counteract IP protection.

CopEvaAtt (Copyright Evasion Attack [30]): To evade the legitimate owner's verification, the attacker attempts to construct a detector to detect whether the queried sample is a clean sample or possibly a fingerprinting sample. The attacker deliberately returns a random label if the detector detects any fingerprinting sample.

Evaluation Metrics: We analyzed the performance of our method by measuring the following metrics. (1) Fidelity: side effects on the main classification tasks, i.e., the global model accuracy (GMC). (2) Validity: whether ownership of the model can be successfully verified, i.e., detector accuracy (DMC). (3) Robustness: resistance to attacks, i.e., detector accuracy after being attacked (DMC_{Att}). The evaluation results regarding GMC, DMC, and DMC_{Att} are all shown in percentage in this section.

Threshold α : We set the threshold α mainly based on the statistics of two experiments. 1) Counting the DMC of the property rights model and the DMC_{Att} in the face of various attacks. 2) Using different training strategies (including federated

learning and single-machine training), the same dataset and model training to get a non-IP model, i.e., without making an IP statement on it, and statistics of its DMC and DMC_{Att} .

14.7 Evaluation and Analysis

In this section, we assess the performance of our method by answering the following three research questions (RQs).

- **RQ1:** Can our method effectively be used to claim the ownership of a given DNN model?
- **RQ2:** Regarding the fidelity of our method, what is its impact on main task performance?

14.7.1 RQ1: Validity

In this section, we evaluate the effectiveness of our method. The purpose of this chapter is to measure the success of the validation of the properties of the target models from the point of view of preserving our approach. We test DMC between IP models and non-IP models (i.e., models that are not ours). In this case, we use the same training data and the same model structure as the IP models. We validate these models using the M detector and estimate the final DMC.

The experimental results show that this detector is effective in recognizing fingerprint features and predicting them as predefined markers with high accuracy. As shown in Table 14.1, the accuracy of all non-IP models is only 8.80%–18.40%, which is about the same as that of random guessing, i.e., our method does not falsely claim ownership of non-IP models. In contrast, it is encouraging to note that our method achieves 100% accuracy for models with declared IPs. This is due to the fact that each declared IP model has a unique model fingerprint that can be effectively predicted during subsequent validation.

Answer to RQ1: Our method effectively verifies the ownership of the target model and achieves 100% DMC without falsely claiming the ownership of the non-IP model.

Table 14.1 The accuracy of detector for nine different models with three datasets. High DMC indicates the ownership of the model is verified

Model	LeNet-1	LeNet-4	LeNet-5	VGG-11	VGG-13	VGG-16	ResNet-18	ResNet-34	ResNet-50
IP	95.56	95.60	95.45	95.68	95.90	95.76	95.98	96.39	95.88
Non-IP	17.15	14.77	12.76	16.71	13.58	18.09	10.12	10.35	8.90

14.7.2 RQ2: Fidelity

In this section, we evaluate the potential side effects of our method on the main task performance.

Fidelity requires the implementation of IP protection without significantly affecting the primary task. We tested on three datasets with nine models to compare the difference in global model accuracy between normal federated training and federated training using our framework.

The experiments demonstrate that our method has excellent fidelity. The results are shown in Table 14.2, the global model for protecting intellectual property is usually matched to the accuracy of the main task of the learned global model. This is because the learning processes of our method and the primary task are independent of each other. Our method does not actively change the parameters of the global model. It only uses the inputs and outputs of the model to achieve the IP protection goal.

Answer to RQ2: Our method is independent of the main task training, thus it has a strong fidelity and does not have any side effects on the main task accuracy.

14.7.3 RQ3: Robustness

In this section, we use FtuningAtt, PruningAtt, and AdaptiveAtt to evaluate the robustness of our method, and also illustrate the CopEvaAtt.

FtuningAtt. FtuningAtt is actually a generalized strategy that uses 10% of the test set data to tune the trained model and measure the robustness of our method. As can be seen from Table 14.3, even after 100 calendar hours, our method is still the most accurate of all models (only in the worst case it only drops by 19.50%). This is

Table 14.2 The accuracy of the global model with IP protection and the global model for normal federated training

Model	LeNet-1	LeNet-4	LeNet-5	VGG-11	VGG-13	VGG-16	ResNet-18	ResNet-34	ResNet-50
IP	94.20	95.51	95.70	83.70	85.88	86.16	47.10	48.03	52.71
Non-IP	94.20	95.32	95.60	83.73	85.70	86.16	47.10	48.13	52.71

Table 14.3 The change in copyright verification accuracy in terms of DMC after performing fine-tuning using a 10% test set for the nine models

Epoch	MNIST			CIFAR10			CIFAR100		
	LeNet-1	LeNet-4	LeNet-5	VGG-11	VGG-13	VGG-16	ResNet-18	ResNet-34	ResNet-50
0	100.00	100.00	100.00	97.40	93.50	88.31	95.11	89.41	86.98
25	100.00	100.00	100.00	98.05	94.15	90.25	92.45	92.61	89.53
50	100.00	100.00	100.00	98.05	92.85	90.90	87.14	84.70	94.64
75	100.00	100.00	100.00	98.05	92.20	90.90	90.96	86.57	91.29
100	100.00	100.00	100.00	98.05	92.85	90.90	86.96	87.12	87.94

probably due to the fact that FtuningAtt does not lead to significant changes in the model weights. This modification does not have significant side effects on the main task and does not lead to a completely different model.

PruningAtt. It is well known that DNNs have many layers and many parameters, and there may be redundant parameters. Therefore, a plagiarist can use the PruningAtt algorithm to cut down the redundant parameters and obtain a new model that is different from the original model but with comparable accuracy. Using the PruningAtt algorithm, the percentage of parameters with the smallest absolute value is set to 0. In our experiments, the parameters were compressed by 30%–90% in 10% intervals. We then used the original test data to evaluate the accuracy of the model and to determine its impact on the original functionality of the model as well as its impact on our methodology. Ideally, a plagiarist would want to prune the stolen model but still maintain its performance.

Observation of the experimental results in Fig. 14.3. It can be found that the performance of the main task is seriously affected as the PruningAtt pruning rate increases. The verification accuracy of the corresponding detectors also decreases gradually, but they have the verification capability if the IP model is still accurate.

The overall model is also found to be worthless if the DMC of some detectors is below the threshold. For example, for the three models in the CIFAR-10 dataset, if the global model is pruned by 90%, the DMC decreases to 10%, making it impossible to determine features. However, the accuracy of the global model is also reduced to 10% at the same time, so there is no protection (Fig. 14.4).

CollabAtt. We design two experimental scenarios to satisfy the inter-collaboration of malicious parties.

CollabAtt_{up}. Upstream malicious client collaboration. Malicious clients participating in FL training can collaborate to eliminate fingerprints from the model and use local training data for hostile retraining attacks.

CollabAtt_{updo}. Collaborative Attacks by Upstream and Downstream Malicious Parties. In this model, upstream malicious clients utilize local training data for adversarial retraining, while downstream malicious clients perform attacks (e.g., branch pruning and fine-tuning) against the learned global model, resulting in coordinated attacks by both upstream and downstream attackers.

We test the effect of two collaborative attacks on different proportions of malicious clients. The experimental results are shown in Fig. 14.5. We find that our method remains robust in the face of collaborative attacks, the main threat to IP verification comes from downstream attackers. *CollabAtt_{up}* cannot effectively affect the verification accuracy of the detector *M*. Facing *CollabAtt_{ftun}* and *CollabAtt_{prun}*, the detector *M* verification accuracy has a certain decrease, but does not affect the effective judgment of the detector *M*. In addition, we investigate the impact of the number of malicious clients on our approach and find that the number of malicious clients does not affect our global IP. Unlike watermark embedding techniques, most malicious clients are effective in removing embedded watermarks from the global model. However, in our framework, all attacks by clients only affect the global model, and the detector is dynamically trained to protect the IP.

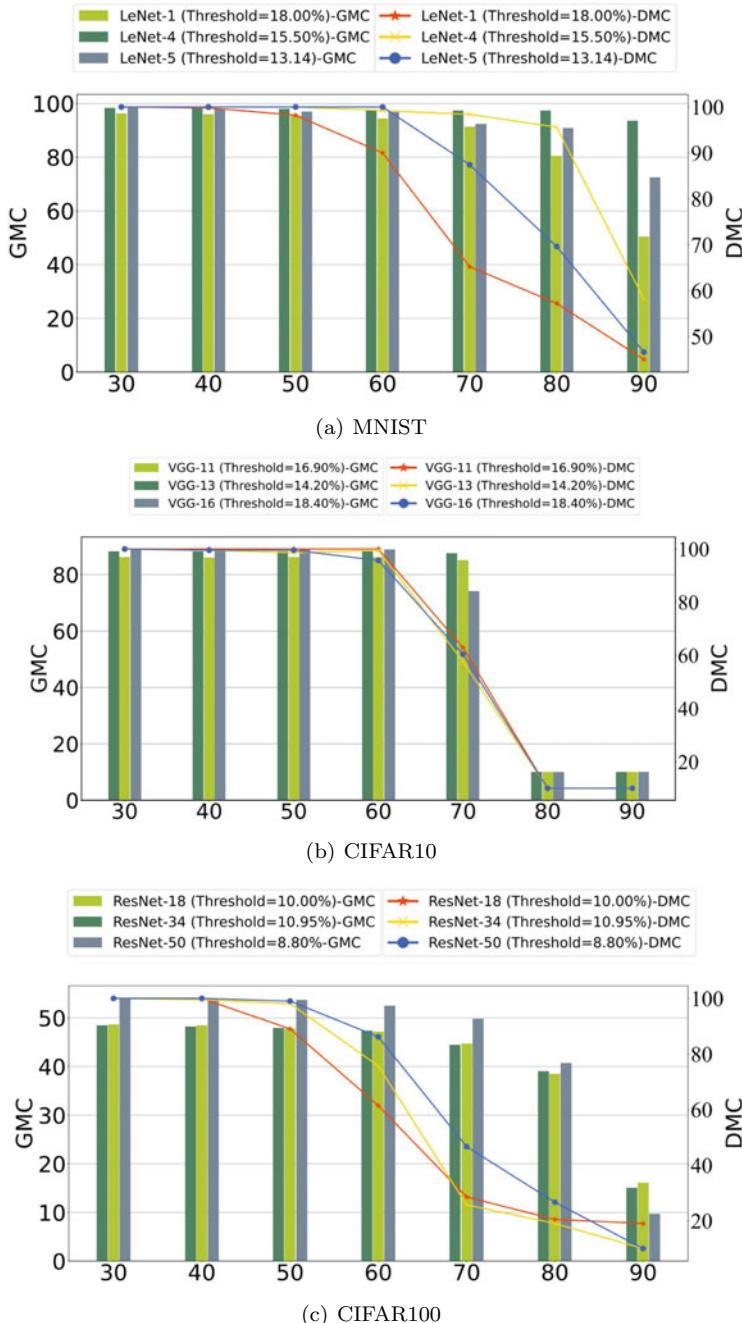


Fig. 14.3 The change in copyright verification accuracy in terms of DMC after performing fine-tuning using a 10% test set for the nine models

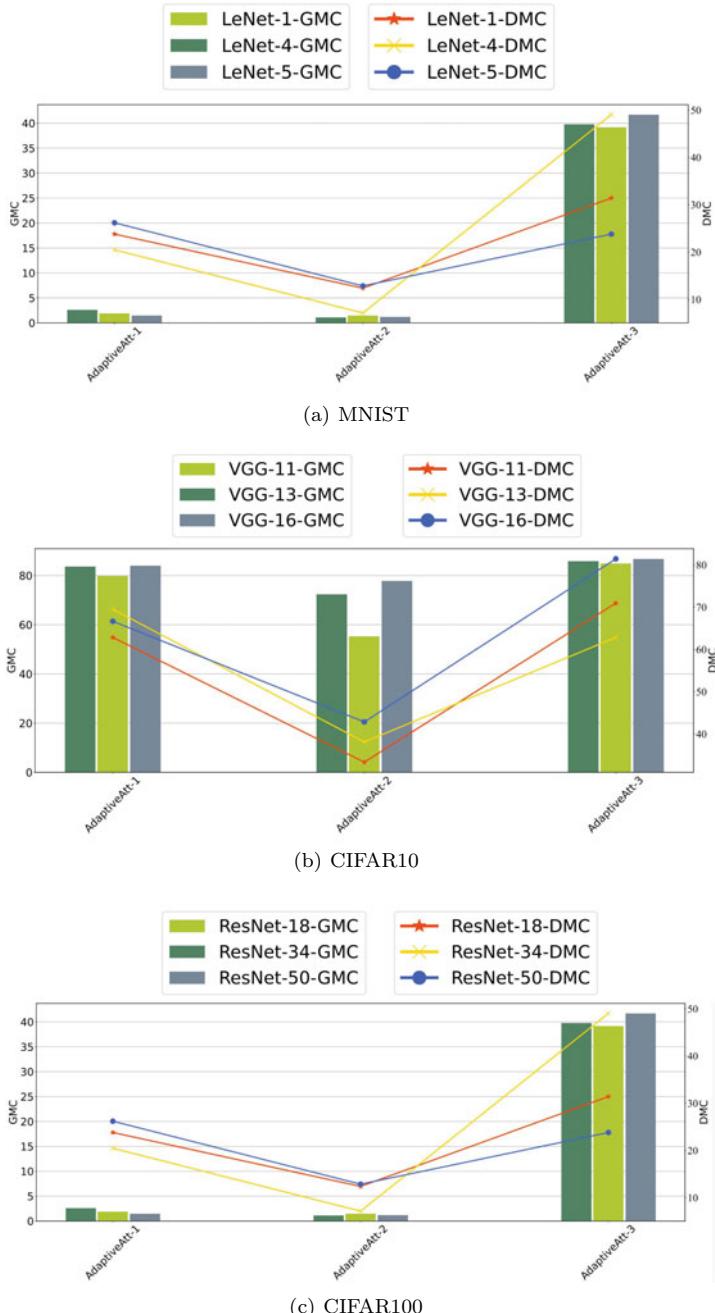


Fig. 14.4 Changes in global model accuracy and detector model accuracy after performing PruningAtt on the nine models

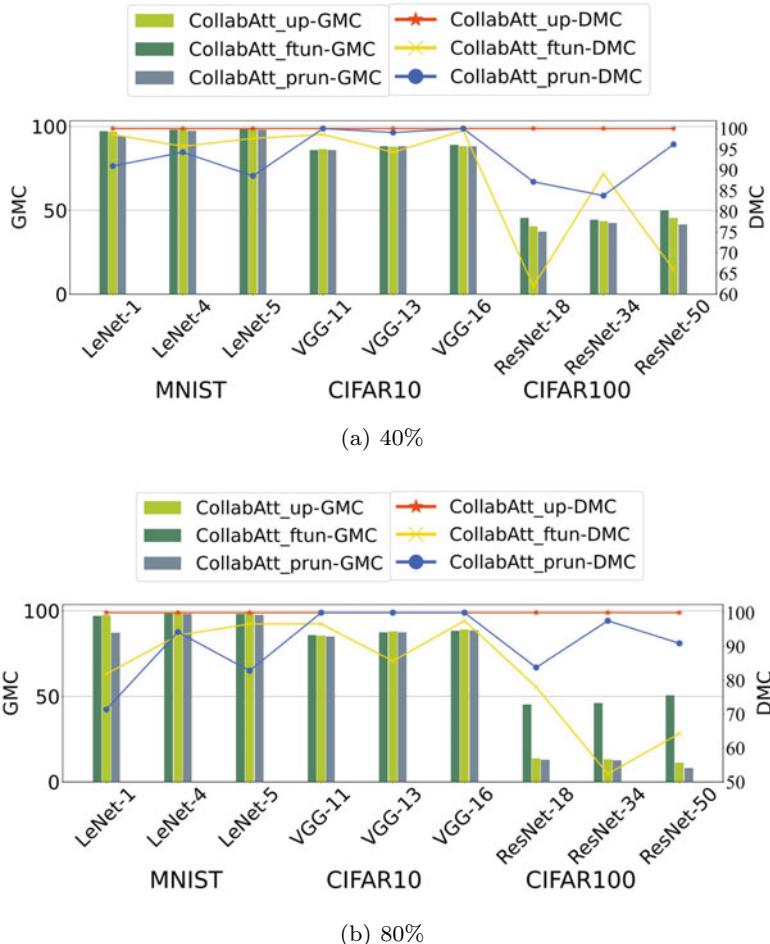


Fig. 14.5 The experimental results of CollabAtt where attackers have different ways of collaboration

AdaptiveAtt. We designed AdaptiveAtt with different levels of adversary capability to test the robustness of IP verification.

AdaptiveAtt-1. In the FL scenario, it is assumed that the attacker has 10% of the original training samples because it is difficult to obtain all the training data from the client. The attacker can use the original training samples to generate adversarial samples and retrain the original model adversarially to remove fingerprints.

AdaptiveAtt-2. All key samples are available to the attacker. The adversarial samples are generated using the key samples. The original model is then adversarial retrained to remove fingerprints.

AdaptiveAtt-3. The attacker has AdaptiveAtt-3, which includes AdaptiveAtt-1 and AdaptiveAtt-2, and first antagonistically retrains the key samples to remove fingerprints and then retrains using the original training samples.

We tested on nine models, and the experimental results are shown in Fig. 14.4. The results show that the attacker using AdaptiveAtt-1 is unable to remove fingerprint matching from the model. Meanwhile, using AdaptiveAtt-2 can effectively remove fingerprints from the model, but it has a significant impact on the prediction performance of the main task, thus significantly degrading the performance of the main task and rendering the attack ineffective; with AdaptiveAtt-3, the accuracy of the model is improved after the attacker adversarial retraining of the original training samples using AdaptiveAtt-2 improved. However, the model's fingerprint can still be removed. However, the fingerprints of the model can still be validated.

CopEvaAtt. To avoid verification by the rightful owner, an attacker can build a detector to detect whether the queried sample is a clean sample or a fingerprint sample. Once the detector determines that the instance being queried is a fingerprint sample, the attacker can consciously return a random label.

However, each client's data is stored locally and is not shared during the federated learning process. It is also uncertain whether the data are independent and identically distributed (IID) across clients. It is not advisable for malicious clients to try to defend against CopEvaAtt's copyright verification. This detection method is costly for the attacker and difficult to implement in practice. The success rate of copyright verification defenses will be accompanied by more false positive sacrifices, which will reduce the effectiveness of the attacker's model for other users.

Answer to RQ3: Our method is robust against model modification attacks including FtuningAtt and PruningAtt. In other words, it is also effective against AdaptiveAtts, which either maintains high validation efficiency or invalidates the model theft objective by reducing the accuracy to undesirable levels.

14.8 Conclusion

This chapter describes a model fingerprint-based FL copyright protection method. The method utilizes the features of models to generate model fingerprints with adaptive extensions. Based on the distribution characteristics of the model fingerprints, a detector is trained to determine the ownership of suspicious models. Our method does not require training data specifically designed for FL scenarios. In addition, our method requires only the inputs and outputs of suspicious models to verify authorship, which is suitable for the real-world scenario where model owners do not have access to the stolen models. In this chapter, we evaluate our method on various datasets and show that it achieves excellent performance in terms of fidelity, efficiency, and robustness.

References

1. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 54, pp. 1273–1282. PMLR (2017)
2. McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging. CoRR [abs/1602.05629](https://arxiv.org/abs/abs/1602.05629) (2016)
3. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. **10**(2), 12:1–12:19 (2019)
4. Shangi, G.: A federated learning based approach for loan defaults prediction. In: Fatta, G.D., Sheng, V.S., Cuzzocrea, A., Zaniolo, C., Wu, X. (eds.) 20th International Conference on Data Mining Workshops, pp. 362–368. IEEE (2020)
5. Kuo, T., Pham, A.: Detecting model misconducts in decentralized healthcare federated learning. Int. J. Med. Inf. **158**, 104658 (2022)
6. Wahab, O.A., Rjoub, G., Bentahar, J., Cohen, R.: Federated against the cold: a trust-based federated learning approach to counter the cold start problem in recommendation systems. Inf. Sci. **601**, 189–206 (2022)
7. Uchida, Y., Nagai, Y., Sakazawa, S., Satoh, S.: Embedding watermarks into deep neural networks. In: Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval, pp. 269–277. ACM (2017)
8. Vybornova, Y.D.: Method for copyright protection of deep neural networks using digital watermarking. In: Fourteenth International Conference on Machine Vision. SPIE Proceedings, vol. 12084, p. 1208412. SPIE (2021)
9. Li, M., Zhong, Q., Zhang, L.Y., Du, Y., Zhang, J., Xiang, Y.: Protecting the intellectual property of deep neural networks with watermarking: The frequency domain approach. In: 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 402–409. IEEE (2020)
10. Zhao, J., Hu, Q., Liu, G., Ma, X., Chen, F., Hassan, M.M.: AFA: adversarial fingerprinting authentication for deep neural networks. Comput. Commun. **150**, 488–497 (2020)
11. Lukas, N., Zhang, Y., Kerschbaum, F.: Deep neural network fingerprinting by conferrable adversarial examples. In: 9th International Conference on Learning Representations. OpenReview.net (2021)
12. Luo, B., Liu, Y., Wei, L., Xu, Q.: Towards imperceptible and robust adversarial example attacks against neural networks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, pp. 1652–1659. AAAI Press (2018)
13. Huang, S.H., Papernot, N., Goodfellow, I.J., Duan, Y., Abbeel, P.: Adversarial attacks on neural network policies. In: 5th International Conference on Learning Representations. OpenReview.net (2017)
14. Wei, L., Luo, B., Li, Y., Liu, Y., Xu, Q.: I know what you see: Power side-channel attack on convolutional neural network accelerators. In: Proceedings of the 34th Annual Computer Security Applications Conference, pp. 393–406. ACM (2018)
15. Li, Z., Hu, C., Zhang, Y., Guo, S.: How to prove your model belongs to you: a blind-watermark based framework to protect intellectual property of DNN. In: Proceedings of the 35th Annual Computer Security Applications Conference, pp. 126–137. ACM (2019)
16. Tekgul, B.G.A., Xia, Y., Marchal, S., Asokan, N.: WAFFLE: watermarking in federated learning. In: 40th International Symposium on Reliable Distributed Systems, pp. 310–320. IEEE (2021)
17. Fan, L., Li, B., Gu, H., Li, J., Yang, Q.: Fedipr: Ownership verification for federated deep neural network models. CoRR [abs/2109.13236](https://arxiv.org/abs/abs/2109.13236) (2021)
18. Li, F., Wang, S., Liew, A.W.: Watermarking protocol for deep neural network ownership regulation in federated learning. In: IEEE International Conference on Multimedia and Expo Workshops, pp. 1–4. IEEE (2022)

19. Wu, W.: Towards efficient horizontal federated learning. Ph.D. thesis, University of Warwick, Coventry, UK (2021)
20. Rouhani, B.D., Chen, H., Koushanfar, F.: Deepsigns: an end-to-end watermarking framework for ownership protection of deep neural networks. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 485–497. ACM (2019)
21. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. CoRR [abs/1608.04644](https://arxiv.org/abs/1608.04644) (2016)
22. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations. OpenReview.net (2018)
23. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
24. Ayi, M., El-Sharkawy, M.: Rmnv2: Reduced mobilenet V2 for CIFAR10. In: 10th Annual Computing and Communication Workshop and Conference, pp. 287–292. IEEE (2020)
25. Singla, S., Singla, S., Feizi, S.: Improved deterministic l2 robustness on CIFAR-10 and CIFAR-100. In: The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25–29, 2022. OpenReview.net (2022)
26. El-Sawy, A., El-Bakry, H.M., Loey, M.: CNN for handwritten arabic digits recognition based on lenet-5. In: Proceedings of the International Conference on Advanced Intelligent Systems and Informatics. Advances in Intelligent Systems and Computing, vol. 533, pp. 566–575 (2016)
27. Chen, H.: Reliable and efficient distributed machine learning. Ph.D. thesis, Royal Institute of Technology, Stockholm, Sweden (2022)
28. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778. IEEE Computer Society (2016)
29. Shafieinejad, M., Lukas, N., Wang, J., Li, X., Kerschbaum, F.: On the robustness of backdoor-based watermarking in deep neural networks. In: IH&MMSec '21: ACM Workshop on Information Hiding and Multimedia Security, pp. 177–188. ACM (2021)
30. Hitaj, D., Hitaj, B., Mancini, L.V.: Evasion attacks against watermarking techniques found in mlaas systems. In: 6th International Conference on Software Defined Systems, SDS 2019, Rome, Italy, June 10–13, 2019, pp. 55–63. IEEE (2019)

Chapter 15

Guard the Vertical Federated Graph

Learning from Property Inference Attack



15.1 Introduction

Recently, graph neural networks (GNNs) have attracted tremendous attention from both academia and industry [1, 2], due to their powerful representation capability for graph-structured data. In practice, GNNs face challenges in utilizing large-scale labeled data to train deep models due to data isolation issue [3, 4], i.e., massive data are owned by different institutes. For example, a bank might have the “income” information of a user, while another lending institution has the “lend records” information of the user.

Consequently, to train accurate GNNs while keeping local data privacy protected for each institute, multiple institutes usually cooperate with each other by contributing data characterizing different aspects of the samples. Thus, vertical federated graph learning (VFGL) is proposed as a solution [5, 6]. Each client in VFGL contributes their local embeddings, and the server combines all local embeddings to make the final prediction [6, 7]. Several VFGL frameworks are developed for real-world applications, e.g., vertical federated GNN (VFGNN) framework with differential privacy from ant group [6], and node-level federated GNN (FedGraphNN) platform from Tencent AI [7].

Despite its practicability, the privacy of VFGL is seldom studied. In this chapter, we carry out the security evaluation of VFGL and dive into its data leakage risks. In particular, we introduce a property inference attack (PiAttack) that allows a malicious server to easily gain sensitive information [8, 9] about a specific user from a client. Our empirical results show that even with only 1% of samples as the auxiliary data via crawling [10], hacking [11], the property inference attack could achieve an accuracy of 0.85 in inferring the data of the client.

There are currently numerous privacy-preserving methods applied to various data structures. However, these methods cannot be applied directly to GNN. Firstly, graph data is more sparse in terms of information about individual samples but more closely correlated between samples than previous data. This makes existing methods of privacy protection reduce the utility of graph data. Secondly, the complexity of the

downstream tasks of graph data makes it difficult to balance the performance of the main task with the privacy-preserving performance of existing privacy-preserving methods. Therefore, privacy leakage on GNNs still needs to be explored.

Further, several privacy-preserving methods have been proposed to address the privacy leakage problem on GNNs, roughly cast into two categories, but there are still shortcomings for VFGL. The first line of work [12, 13] utilized adversarial training techniques to hide the private information during the training. Unfortunately, these methods cannot be applied to VFGL since they are difficult to solve non-differentiable and constrained convex optimization in VFGL. Another line of work [14, 15] explored property protection by injecting perturbations. However, they fail to balance privacy protection and maintain primary task performance at the same time. Therefore, we need an alternative privacy-preserving method to defend the PiAttack in VFGL. In summary, there are two key challenges: 1) How to solve the non-differentiable and constrained convex optimization during each iteration of training in VFGL? 2) How to maintain the primary task performance while keeping the defense robustness?

To address these challenges, we propose a novel defense method against PiAttack in VFGL. Our naive idea is that the victim clients inject specific noise into the embeddings uploaded to the server for protecting the data. Therefore, it is able to implement the privacy preservation locally without the need for non-differential and constrained convex optimization in VFGL. In addition, to balance privacy protection and primary task performance, our method designs two optimization objectives, and the gradient descent optimization method can make the VFGL converge. Specifically, it protects the privacy property by increasing the mutual information between the embeddings of added noises and the fake labels to break the mapping between the noisy embeddings and the sensitive properties. Then, our method adopts the norm to constrain the size of the added noises to guarantee the primary task performance. Our empirical evaluations show that our method outperforms the state-of-the-art methods on six real-world datasets for two common tasks. Moreover, our method can be generalized to different practical VFGL frameworks. We also engage in visualization techniques t-SNE to explain why our method works.

The contributions of this work are summarized as follows:

- To the best of our knowledge, this is the first work to introduce a general PiAttack targeting the inference of the private property for VFGL.
- To defend against the PiAttack on VFGL, we propose a perturbation-based defense method, our method, which balances the privacy and accuracy. By introducing the noise generator module in our method, the entropy between noisy embeddings and private properties is increased to protect privacy.
- Extensive experiments on various datasets and models have been conducted, and results show that our method outperforms the state-of-the-art privacy-preserving strategies against PiAttack. Besides, the experiments show that our method can be practically applied to industrial scenarios, e.g., Amazon product VFGL platforms.

15.2 Related Work

In this section, we briefly review the literature of VFGL methods, inference attack on VFL, and privacy-preserving methods on graph.

15.2.1 *Inference Attack on VFL*

Exchanging embeddings during VFL will lead to potential privacy leakage [16–18]. According to the type of leakage data, inference attack in VFL can be categorized as label inference attack, property inference attack, and data reconstruction attack.

Label inference attack usually uses communicated gradients to identify an effective method to uncover the label of client. In [16], it shows that using the norm of the communicated gradients between the training clients, to a large extent, can reveal the ground-truth labels from the clients on VFL. Property inference attack is one of the most common attack on VFL, by inferring private property of the target example from the embedding. Previous work [17] has shown that a malicious adversary uses a generative regression network in the model inference stage and combines its data to steal other examples' private features. Data reconstruction attack is one of the most serious privacy leakages on VFL. The adversary tries to directly recover the original training data of the victim on VFL. In [18], it shows that reverse sum attack and reverse multiplication attack can cause the original data leakage of VFL, even though the adversary has little prior knowledge about the data distribution of the target example. To the best of our knowledge, there is still no privacy leakage attack on VFGL.

15.2.2 *Privacy-Preserving Methods*

Existing methods for defending against property inference attack can be roughly cast into two categories, one is adversarial training, and the other is differential privacy.

Adversarial training: Liao et al. [12] proposed a GNN encoder based on a min–max game to filter private information from the representations. Li et al. [13] presented a graph adversarial training framework that combines the adversarial autoencoder with purging mechanisms for privacy preservation. However, these methods are only suitable for centralized GNN scenarios. Because clients in VFGL need to iteratively exchange gradient information, however, their methods are difficult to solve for non-differential and constrained convex optimization. Therefore, it cannot be applied to defend VFGL against PiAttack.

Differential privacy: There are several differential privacy methods for protecting data information [19–22]. Baek et al. [20] developed a differential privacy mechanism robust to user dropouts by dynamically calibrating noise with account of the dropout

rate. Yin et al. [22] designed a local Bayesian differential privacy to improve the adaptability of different distributed datasets. However, differential privacy methods mainly aim at defending the membership inference attack, which strives to find randomized algorithms so that the adversary cannot infer whether an individual appears in the training data. Moreover, differential privacy tends to focus on the preservation of privacy at the expense of primary task performance. Therefore, it might not be suitable for defense against property inference attack in VFGL.

15.3 Methodology

To address the constraints, we propose a novel perturbation-based defense method, our method, as shown in Fig. 15.1. To solve the first constraint, we build a fake relation mapping strategy that perturbs the correlation between private data and its embeddings to maintain data privacy. To solve the second constraint, we build a feature space movement constraint. We make the node embeddings of different classes move as little as possible in the feature space.

The fake relation mapping strategy consists of two key modules, namely fake labels generation module and noise generation module. The feature space movement constraint strategy is composed of a key feature space drift constraint. More specifically, we first generate evenly distributed fake labels through the fake labels generation module according to the classes of private property. Then we generate specific noise with feature space drift constraint for the embeddings through the noise generation module. Finally, we add noise to the original embeddings to get the final embeddings that can be uploaded to the server. In the following part, we will introduce the two key strategies, respectively.

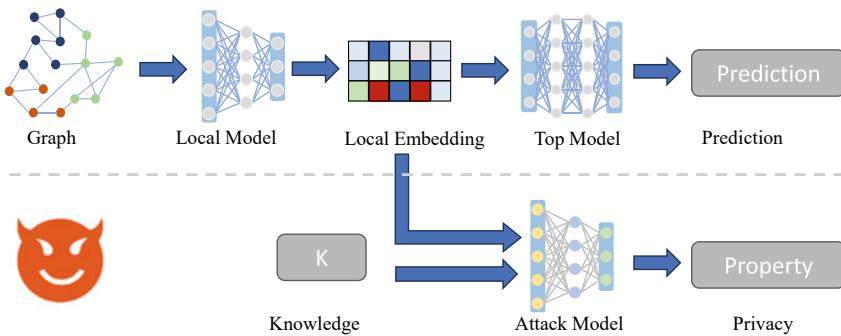


Fig. 15.1 The framework of PiAttack in VFGL, which the adversary can steal the private data with an attack classifier

15.3.1 Details of Strategies

The principle of the false relationship mapping strategy is that the clients randomly shift the node embeddings in the feature space into several classes by adding noise, and each class is re-assigned a new property category, thereby establishing the false node embeddings and labels mapping relationship. For example, 900 balanced-distributed node embeddings with property “income” are randomly divided into three categories, each of which is assigned a false label. The new class contains 100 “high income” node embeddings, 100 “middle income” node embeddings, and 100 “low income” node embeddings. By adding noise, the node embeddings drift in the feature space to the category of false labels.

The fake relation mapping strategy is divided into three steps. The first step is to determine the number of categories generated by fake labels. The second step is to generate node embeddings and fake label pairs. The third step is to establish the mapping relationship between node embeddings and fake labels.

15.3.1.1 Step 1: Choose the Number of Fake Labels’ Classes

The number of categories of fake labels is usually the number of categories of protected privacy properties. This facilitates the defender to generate uniformly distributed fake labels to maximize the entropy of node embeddings and true properties. For example, for the “income” property, the category is set to 3. In reality, the number of categories of protected privacy properties is usually public information, for example, the category number of gender is 2, which is easily obtained by defenders.

15.3.1.2 Step 2: Generate the Pairs of Fake Labels and Embeddings

We generate random labels based on the number of categories, where each label category is assigned the same number of node embeddings. Given the number J of the classes of private property, fake label generation randomly allocates the same number of examples to each class of property. The corresponding relationship between fake labels Y^f and examples $\{x_i\}_{i=1}^Z$ is as

$$Y^f = \mathcal{S}_f \left(\sum_{j=1}^J \sum_{i=1}^{Z/J} (x_i, j) \right) \quad (15.1)$$

where Z is the number of examples and $\mathcal{S}_f(\cdot)$ is the random shuffle function.

15.3.1.3 Step 3: Generate Noise

The purpose of the noise generator is to generate specific noise that the feature space of the node embedding changes, so that the node embeddings are randomly mapped to the fake label category. The noisy node embeddings are not subject to PiAttack. We use the cross-entropy loss (\mathcal{C}_E) to guide the noise generation, the loss is given by

$$\mathcal{L}_{\text{privacy}} = \mathcal{C}_E(E_c, Y^f) \quad (15.2)$$

where the noisy embeddings $E_c = E_t + f_{\theta_n}(E_t)$.

To take account of constraint 2, we propose a norm constraint, making the noisy embeddings E_c retain as much information as possible from the raw data:

$$\mathcal{L}_{\text{utility}} = d(E_c, E_t) \quad (15.3)$$

where $D(\cdot)$ is a distance function, e.g., the mean squared error.

The overall objective function of training the noise generator is thus given by

$$\text{Loss} = \mathcal{L}_{\text{utility}} + \lambda \mathcal{L}_{\text{privacy}} \quad (15.4)$$

where λ is hyperparameter that control the utility and privacy. In the training process, we update the noise generator f_θ until they converge.

Our noise generator usually consists of a multi-layer fully connected neural network with ReLU activation function. The loss function of the noise generator is mainly composed of two parts, one part is used to protect data privacy; the other part is used to ensure the performance of the main task, which will be analyzed and introduced below.

We put random noise into the noise generator model. After the forward propagation is completed, the model output results and fake labels are used to calculate the loss according to Eq. 15.4, and then the backpropagation is performed to calculate the gradient of the generator model, and the gradient descent optimization algorithm is used to update the model parameters. Once the generator model is trained, it can generate special noise for clients.

15.3.2 Theoretical Discussion

In this subsection, we prove that the uniform distribution leads to maximized entropy.

Theorem 15.1 *Entropy is maximized if public embeddings distribution p is uniform, which is expressed as*

$$H(X) \leq \log(|\mathcal{A}_X|) \text{ iff } p = 1/|\mathcal{A}_X|. \quad (15.5)$$

where $H(X)$ is the entropy of X , \mathcal{A}_X is the public embedding set, “iff” means “if and only if”. $|\mathcal{A}_X|$ denotes the number of elements in \mathcal{A}_X . If all sample embeddings in the set \mathcal{A}_X are picked with equal probability $1/m$ (m being cardinality of the set \mathcal{A}_X), then the entropy increases. The method achieves great privacy-preserving performance.

Proof We consider a probability density function on $\{x_1, \dots, x_n\}$. Entropy is a continuous function of the n -tuples (p_1, \dots, p_n) , and these points lie in a compact subset of \mathbb{R}^n . Suppose the $p_1 < p_2$. For small positive ε , we have $p_1 + \varepsilon < p_2 - \varepsilon$. The entropy of $\{p_1 + \varepsilon, p_2 - \varepsilon, \dots, p_n\}$ minus the entropy of $\{p_1, p_2, \dots, p_n\}$ equals L_A .

$$\begin{aligned} L_A = & -p_1 \log\left(\frac{p_1 + \varepsilon}{p_1}\right) - \varepsilon \log(p_1 + \varepsilon) \\ & -p_2 \log\left(\frac{p_2 - \varepsilon}{p_2}\right) + \varepsilon \log(p_2 - \varepsilon) \end{aligned} \quad (15.6)$$

Rewrite Eq. 15.6 as

$$\begin{aligned} & -p_1 \log\left(1 + \frac{\varepsilon}{p_1}\right) - \varepsilon \left(\log p_1 + \log\left(1 + \frac{\varepsilon}{p_1}\right)\right) \\ & -p_2 \log\left(1 - \frac{\varepsilon}{p_2}\right) + \varepsilon \left(\log p_2 + \log\left(1 - \frac{\varepsilon}{p_2}\right)\right) \end{aligned} \quad (15.7)$$

Recalling that $\log(1 + x) = x + O(x^2)$ for small x , Eq. (15.7) is

$$\varepsilon \log(p_2/p_1) + O(\varepsilon^2) \quad (15.8)$$

which is positive when ε is small enough since $p_1 < p_2$.

15.4 Experiments

This section demonstrates the feasibility of PiAttack and applies our method to different VFGL downstream tasks by comparing it with current state-of-the-art methods. The main research questions (RQ) are:

- **RQ1:** Does PiAttack have strong attack performance on VFGL? Does it work for different tasks?
- **RQ2:** Does our method achieve the state-of-the-art privacy-preserving performance, taking into account the performance of the primary task?
- **RQ3:** How stable is our method defense, such as multi-client setting, different model architectures, and different hyperparameters?

15.4.1 Datasets

We select two types of datasets for two downstream tasks of VFGL, i.e., including citation network (Cora, Citeseer and Pubmed [23, 24]) for link prediction task, and social network (Rochester and Yale [13]) for node classification task. The number of classes represents the granularity of privacy properties. To fit these datasets for the scenario of VFGL with two clients, each feature matrix is cut in half with each client holding a half, which does not overlap.

15.4.2 Models

For the local model of the clients, we consider the six most popular GNN networks, such as graph convolutional network (GCN) [25], simplifying graph convolutional networks (SGC) [26], graph attention network (GAT) [27], graph regularization neural network (GNAE) [28], and variational graph auto-encoders (VGAE) [29]. The first four local models learn node embeddings for both node classification and link prediction, while the latter two local models learn node embeddings for only link prediction. The output dimension of the local model is 128. For the top model, we use a three-layer fully connected neural network.

15.4.3 Defense Baselines

To demonstrate the validity of the proposed defense method, we choose some advanced privacy-preserving methods: random noise (Noisy), differential privacy (DP) [14], random dropout (Dropout) [17], and dimensionality reduction (DR) [30] to defend the PiAttack. No_defense denotes the original VFGL model without any defense.

Noisy [8]: Noisy adds Gaussian noise to the original embeddings. The original embedding's representation of superimposed Gaussian noise can provide a privacy guarantee. We set Noisy with different Gaussian noise parameters σ as {1, 5, 10, 15}.

DP [14]: DP injects randomized noise into the graph feature. We implement DP with the different noise parameters as {0.1, 0.3, 0.5, 0.7}, which are similar to the settings used for the existing work[31].

DP_G [32]: DP_G injects geometrically distributed noise into the graph feature. We implement DP with the different noise parameters as {0.1, 0.3, 0.5, 0.7}.

Dropout [17]: Dropout uses for the local model is regarded to defend against inference attack in VFL. We adopt the Dropout with different random dropout rates as {0.1, 0.3, 0.5, 0.7}.

DR [30]: DR removes the private information by decreasing the dimension of local model output. We set the different DRs output dimensions of the local model as {32, 16, 8, 4}.

15.4.4 Metrics

To measure the performance of our method, we utilize five metrics.

- **Primary task accuracy (PTA)** [33]: it corresponds to the fraction of correct predictions made by the VFGL for the node classification task. The higher, the better.

$$PTA = \frac{\text{Number of successful inferred instances}}{\text{Number of inferred instances}} \quad (15.9)$$

- **Primary task AUC (AUC)** [34]: it is frequently used in binary classification tasks. It measures the performance in the link prediction task. If among n independent comparisons, there are n' times that the existing link gets a higher score than the nonexistent link and n'' times they get the same score. The higher, the better.

$$AUC = \frac{n' + 0.5n''}{n} \quad (15.10)$$

- **Privacy inference accuracy (PIA)** [8]: it represents the fraction of correct private property predictions made by the adversary. A lower Inference Accuracy means less privacy leakage.

$$PIA = \frac{\text{Number of successful inferred properties}}{\text{Number of inferred properties}} \quad (15.11)$$

- **Inference macro-F1-score (Macro-F1)** [35]: it is computed by taking the arithmetic mean (unweighted mean) of all the per-class F1-scores. This metric treats all classes equally regardless of their support values, which are combined with precision and recall values.

$$\text{Macro-F1} = \sum_{i=0}^N \frac{2}{N_C} \cdot \left(\frac{\text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}} \right)_{\text{class } i} \quad (15.12)$$

where i is the class index and N_C is the total number of classes.

- **Trade-off value (TOV)** [13]: it measures the trade-off between utility, i.e., the accuracy or AUC of the primary task, and privacy, i.e., inference accuracy of private property. The higher, the better.

$$TOV = \frac{\text{Utility of primary task}}{\text{Inference accuracy}} \quad (15.13)$$

15.4.5 Loss Functions

To demonstrate the effectiveness of our method with different loss functions. We use four different loss functions, the mean absolute error (MAE) [36], mean squared error (MSE) [37], the Kullback–Leibler divergence (KLDiv) [36], and the cross-entropy [29] loss function.

- **Mean absolute error (MAE)** [36]: It calculates the mean absolute error between the predicted label and the true label. The process is as follows:

$$MAE(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^M |\hat{y}^{(i)} - y^{(i)}| \quad (15.14)$$

where y represents the true label, \hat{y} represents the predicted label, and m represents the number of labels.

- **Mean squared error (MSE)** [37]: It calculates the mean squared error between the predicted label and the true label. It is defined as

$$MSE(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^M (\hat{y}^{(i)} - y^{(i)})^2 \quad (15.15)$$

- **Kullback–Leibler divergence (KLDiv)** [36]: It calculates the relative information entropy between the predicted label and the true label. The process is as follows:

$$KLDiv(\hat{y}, y) = \sum_{i=1}^M y^{(i)} * (\log y^{(i)} - \log \hat{y}^{(i)}) \quad (15.16)$$

- **Cross-entropy** [29]: It calculates the cross-entropy between the predicted label and the true label. It is defined as

$$\text{cross entropy}(\hat{y}, y) = - \sum_{i=1}^M y^{(i)} * \log \hat{y}^{(i)} \quad (15.17)$$

15.4.6 Experiment Setup

The classifier of adversary is a four-layer fully connected neural network. The default activation function adopts ReLU. We use the Adam [38] optimizer for the local model and top model parameters. For each experiment, we report the average performance of 5 runs.

15.4.7 RQ1: Attack Effectiveness

We focus on the attack results of PiAttack to demonstrate the risk of private property on VFGL for node classification and link prediction tasks.

Reasons why PiAttack works. The success of property inference attack is mainly due to the strong correlation between nodes' property and client embedding. Specifically, the embedding of the client is obtained by the property of nodes and the links between the nodes through GNN. As shown in Fig. 15.2, when the property of the nodes changes, the client's embedding is changed. Therefore, it is reasonable to infer the property of the node from the client's embedding.

Implementation details. In this experiment, we utilize GCN as the local model to extract embeddings to complete the node classification task and link prediction task. We set dropout rate for all GCN layers, L2 regularization factor for the first GCN layer, and number of hidden units. We train for 100 epochs (without early stopping) with a learning rate of 0.01. Hyperparameters are chosen as follows: 0.5 (dropout rate, first and last layer), 5×10^{-4} (L2 regularization, first layer), 128 (number of units for each hidden layer) and 0.01 (learning rate). Our VFGL framework has two clients.

Results and discussion. Table 15.1 shows the experimental results of PiAttack and random guessing on different downstream tasks and datasets. In Table 15.1, the ordinate represents the inference accuracy, and the abscissa represents the fraction of auxiliary set by the adversary. The random guess is the most straightforward baseline, which varies for different private data.

Generally speaking, PiAttack significantly improves the private data inference accuracy in node classification and link prediction tasks, with its PIA on four datasets almost reaching 0.9. It demonstrates native VFGL's serious risk of private data leakage against PiAttack. For instance, the inference accuracy of PiAttack on the

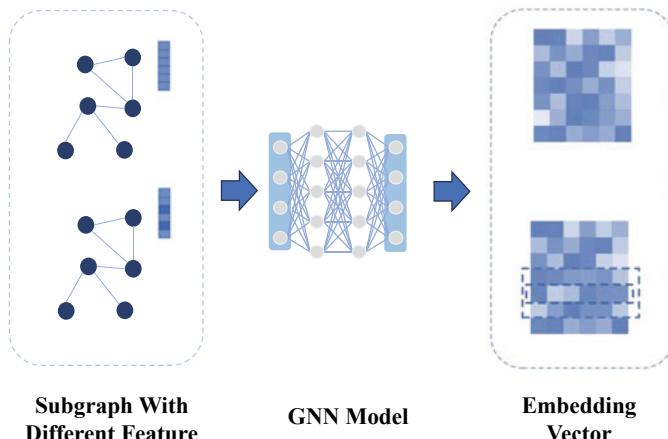


Fig. 15.2 Correlation between user's property and server embedding

Table 15.1 PIA for property inference. The PiAttack and Random method represent the attack with auxiliary dataset and random guessing, respectively

Dataset	Attack methods	Fraction of auxiliary set								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora	PiAttack	0.386	0.408	0.531	0.567	0.621	0.701	0.779	0.779	0.779
	Random guess	0.136	0.136	0.136	0.136	0.136	0.136	0.136	0.136	0.136
Citeseer	PiAttack	0.331	0.347	0.509	0.571	0.669	0.737	0.786	0.789	0.789
	Random guess	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167
Rochester	PiAttack	0.781	0.801	0.789	0.808	0.808	0.811	0.807	0.807	0.807
	Random guess	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173
Yale	PiAttack	0.848	0.867	0.876	0.875	0.886	0.895	0.895	0.895	0.895
	Random guess	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500

Table 15.2 Macro-F1 for property inference. The PiAttack and Random method represent the attack with auxiliary dataset and random guessing, respectively

Dataset	Attack methods	Fraction of auxiliary set								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Cora	PiAttack	0.348	0.378	0.501	0.528	0.588	0.670	0.761	0.761	0.761
	Random guess	0.136	0.136	0.136	0.136	0.136	0.136	0.136	0.136	0.136
Citeseer	PiAttack	0.320	0.329	0.490	0.550	0.663	0.742	0.782	0.782	0.782
	Random guess	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167	0.167
Rochester	PiAttack	0.754	0.761	0.750	0.781	0.781	0.780	0.778	0.778	0.778
	Random guess	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173	0.173
Yale	PiAttack	0.825	0.856	0.865	0.865	0.875	0.885	0.884	0.884	0.885
	Random guess	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500

Rochester dataset almost reaches 0.98, which is almost 0.78 higher than random guess. The above results experimentally show that the PiAttack successfully establishes the mapping relationship between embedding and private data via an auxiliary set (Table 15.2).

The inference accuracy of PiAttack increases with the fraction of the auxiliary set. When the adversary collects more auxiliary sets, VFGL will have a more serious privacy leakage risk. In addition, we note that, when the adversary has only 1% auxiliary data on the Yale dataset, the inference accuracy on “gender” private data reaches around 0.83. As we expected, the smaller J of private data requires fewer auxiliary samples.

15.4.8 RQ2: Defense Effectiveness

We focus on the defense results of our method to measure the effectiveness against PiAttack. Then, we explain the effectiveness of our method with t-SNE visualization technology.

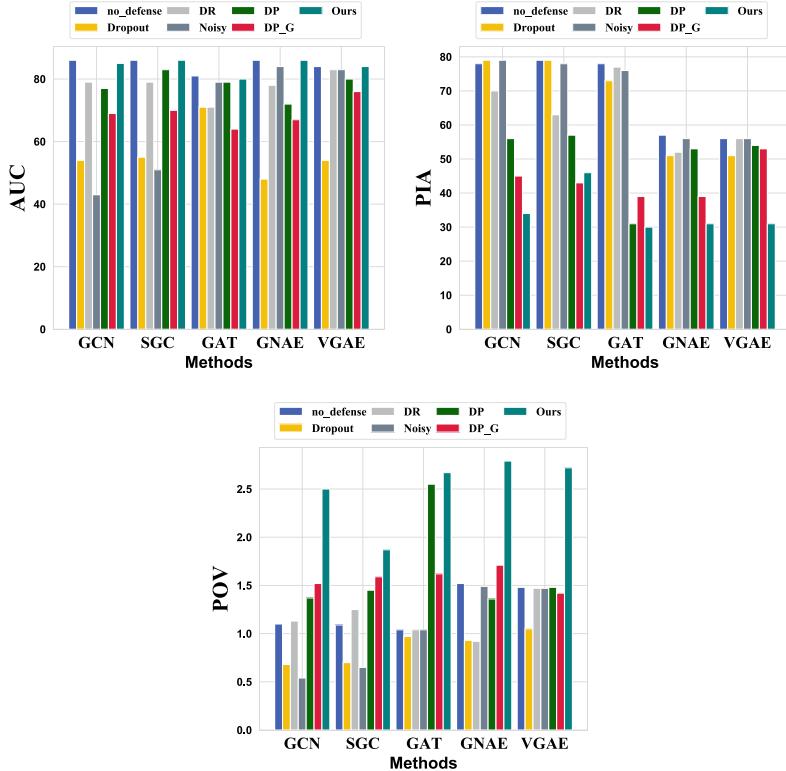


Fig. 15.3 The task accuracy and data privacy of our method and other methods with different GNNs. The higher the accuracy of the primary task, the better utility. The lower inference accuracy, the better privacy preservation. The higher trade-off value, the better balance

15.4.8.1 Our Method Balances the Privacy and Task Performance

Here, we study our method trade-offs of privacy-preserving and task performance to measure the effectiveness.

Implementation details. In this experiment, we set the strongest adversary to launch the PiAttack to steal private data. The adversary's fraction of the auxiliary set reaches: 70%, 70%, 10%, 3%, respectively, on four datasets. our method has only one hyperparameter λ to control the trade-off between data privacy and task accuracy. We set five different $\lambda \{0.1, 0.05, 0.01, 0.005, 0.001\}$ for our method.

Results and discussion. No_Defense represents the privacy leakage of native VFGL without any defense suffering the PiAttack. The lower the inference accuracy and the higher the primary task AUC or accuracy, the better.

Our method achieves the best privacy-utility tradeoff under most experiment settings, i.e., the dots representing our method appear closer to the top right corner compared to other baselines. DP_G achieves a privacy inference accuracy of 0.45

and a main task AUC of 0.72 on the Cora dataset, while our method achieves a lower privacy inference accuracy and better main task performance. This demonstrates that our method can efficiently protect privacy while maintaining the task accuracy in VFGL for both node classification and link prediction tasks.

15.4.9 RQ3: Sensitivity Evaluation

15.4.9.1 Evaluation of Our Method with Different Hyperparameters

Here, we study the impact of hyperparameter on the defense effect of our method.

Results and discussion. As Table 15.3 illustrates, our method balances data privacy and task accuracy. As expected, we observe that with the λ increases, the utility of embeddings decreases monotonically and the privacy preservation increases monotonically. Meanwhile, with the λ increases by a tiny step, the privacy preservation improves significantly, and the utility decreases slightly when $\lambda \in [0, 0.001]$ on the Cora dataset. When $\lambda \in [0.01, 0.1]$, with the λ increases by a tiny step, privacy preservation improves slightly, and the utility drops rapidly. The same conclusion still holds for other datasets. As the intensity of the noise increases, the utility of embedding increases rapidly.

15.4.9.2 Our Method Is Effective on Different Models

We further investigate whether our method applies to the other GNN network models of the clients besides the GCN model. Concretely, we focus on SGC, GAT, GNAE, and VGAE. The experimental results on the Cora dataset are shown in Fig. 15.3.

Implementation details. For a two-layer GAT model, the first layer consists of eight attention heads computing eight features each, followed by an exponential linear unit [39] non-linearity. The second layer is used for classification: a single attention head that computes 128 features, followed by ReLU activation. For coping with the small training set sizes, regularization is liberally applied within the model. During training, we apply L2 regularization with $5 * 10^{-4}$. Furthermore, dropout with 0.5 is applied to both layers' inputs, and the learning rate adopts 0.01. For a two-layer GNAE, the first layer is a linear layer with 128 units, followed by an L2 regularization operation, and the second layer is an aggregation layer. Note that the SGC model, the GCN model, and the VGAE model keep the same hyperparameter settings (Table 15.4).

Results and discussion. We observe from Fig. 15.3 that for other GNN models, our method also performs well in protecting privacy (low inference accuracy) as well as balancing the task accuracy (high primary task accuracy). For example, when the client adopts the GAT model on the Cora dataset, our method reduces the adversary classification accuracy from 0.78 to 0.3, while the accuracy of the primary task only reduces 0.1. Compared with other baselines, our method achieves the

Table 15.3 Impact from different λ on the tradeoff between primary task accuracy and the privacy preservation effectiveness

Dataset	Metric	Lambda			
		0	0.001	0.01	0.1
Cora	AUC	0.900	0.889	0.870	0.859
	PIA	0.899	0.843	0.837	0.828
Citeseer	AUC	0.900	0.889	0.869	0.860
	PIA	0.896	0.827	0.819	0.808

Table 15.4 Impact from different λ on the tradeoff between primary task accuracy and the privacy preservation effectiveness

Dataset	Metric	Lambda			
		0	0.001	0.01	0.1
Rochester	PTA	0.860	0.850	0.830	0.820
	PIA	0.859	0.803	0.797	0.788
Yale	PTA	0.839	0.839	0.828	0.820
	PIA	0.852	0.830	0.799	0.797

best balance between data privacy and task accuracy. This further demonstrates that our method is generally applicable to different GNN network models of the clients. Our method is effective for different models because it only pays attention to the embedding generated by the client and ignores the specific model structure of the client's GNN model.

Moreover, we find that the non-linear activation function in the local model can reduce the private property information. One possible reason is that the non-linear activation function makes the activation value non-zero, which potentially removes the private information. Our insight is that the local model can adopt the non-linear activation function to decrease the risk of privacy leakage.

15.4.9.3 Our Method Is Effective in Multi-client Setting

Here, the capability of our method with the different numbers of clients is further discussed.

Implementation details. The VFGL with a different number of clients is evaluated on the GCN of the Rochester dataset. For measurement, the feature matrix of the clients is evenly divided according to the number of clients, and each client has a part of the feature information without any overlap. We set the same adjacency matrix for each client in VFGL.

Results and discussion. As shown in Table 15.5, the inference accuracy of private property, measured by the accuracy, degrades when the number of clients increases. This is consistent with our expectation, as the client with fewer features leads to

Table 15.5 Multiple clients setting. The higher PTA of the primary task, the better utility. The lower PIA, the better privacy preservation

Methods	Metric	2	3	4	5	6
VFGL	PIA	0.857	0.859	0.868	0.848	0.849
	PTA	0.827	0.831	0.822	0.807	0.808
VFGL with Ours	PIA	0.801	0.657	0.605	0.615	0.675
	PTA	0.494	0.198	0.174	0.182	0.18

Table 15.6 Performance of our method in different loss functions. The lower PIA and F1, the better privacy preservation

Metric	MAE	MSE	Kldiv	CE
PIA	0.151	0.164	0.174	0.150
F1	0.196	0.211	0.199	0.174

embeddings having less relation with the private property. Even if there are six clients in VFGL, our method can reduce the sensitive property inference accuracy of the protected client to about 0.2, which is close to the level of random guesses by the adversary. At the same time, we find that as the number of clients increases, the size of noise is always limited by the regularizer term in our optimization objective (Eq. 15.3). In Table 15.5, the accuracy of our method’s sensitive property inference on multi-client is reduced to below 0.5, especially when there are more than three clients, the accuracy of sensitive property inference reaches around 0.2. This experiment shows that our method can be extended to multiparty scenarios.

15.4.9.4 Our Method Is Effective with Different Loss Function

Here, the capability of our method with the different loss functions is further discussed.

Implementation details. Our method with different loss functions is evaluated on the GCN of the Rochester dataset. In this experiment, we evaluated the performance of our method using four loss functions: MAE, MSE, KLDiv, and cross-entropy, respectively. In addition, the hyperparameter settings for the GCN are the same as for RQ2.

Results and Discussion. As shown in Table 15.6, our method achieves privacy preservation when different loss functions are used. Furthermore, it is worth noting that our method achieves the best performance when using the cross-entropy loss function. This indicates that our method can remain effective when using different loss functions, and that the cross-entropy loss function can achieve the best performance.

15.5 Conclusions

In this work, we first reveal that an adversary can launch a general property inference attack on VFGL to successfully infer the private information of a client. In order to mitigate such threat, we propose our method as privacy protection framework against property inference attack on VFGL. As a privacy protection mechanism, our method achieves a good balance between the task utility, i.e., the primary task accuracy, and the privacy protection, i.e., the inference accuracy. Additionally, our method is general to various GNNs of the clients. We conducted a series of experiments to evaluate our method and found that it outperforms current state-of-the-art methods in VFGL.

References

1. Singh, A., Sharma, S., Gumaste, A., Wang, J.: Grafnet: using graph neural networks to create table-less routers. *IEEE Trans. Netw. Sci. Eng.* **9**(2), 740–754 (2022)
2. Zhang, Z., Li, Y., Dong, H., Gao, H., Jin, Y., Wang, W.: Spectral-based directed graph network for malware detection. *IEEE Trans. Netw. Sci. Eng.* **8**(2), 957–970 (2021)
3. et al., P.K.: Advances and open problems in federated learning. *Found. Trends Mach. Learn.* **14**(1-2), 1–210 (2021)
4. Chen, F., Li, P., Miyazaki, T., Wu, C.: Fedgraph: federated graph learning with intelligent sampling. *IEEE Trans. Parallel Distrib. Syst.* **33**(8), 1775–1786 (2022)
5. Ni, X., Xu, X., Lyu, L., Meng, C., Wang, W.: A vertical federated learning framework for graph convolutional network. *CoRR* **abs/2106.11593** (2021)
6. Zhou, J., Chen, C., Zheng, L., Wu, H., Wu, J., Zheng, X., Wu, B., Liu, Z., Wang, L.: Vertically federated graph neural network for privacy-preserving node classification. *ArXiv preprint arXiv:2005.11903* (2020)
7. He, C., Balasubramanian, K., Ceyani, E., Yang, C., Xie, H., Sun, L., He, L., Yang, L., Yu, P.S., Rong, Y., et al.: Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145* (2021)
8. Zhang, Z., Chen, M., Backes, M., Shen, Y., Zhang, Y.: Inference attacks against graph neural networks. In: Butler, K.R.B., Thomas, K. (eds.) 31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10–12, 2022, pp. 4543–4560. USENIX Association (2022), <https://www.usenix.org/conference/usenixsecurity22/presentation/zhang-zhikun>
9. Duddu, V., Bouvet, A., Shejwalkar, V.: Quantifying privacy leakage in graph embedding. In: Mühlhäuser, M., Polyzos, G.C., Michahelles, F., Guinea, A.S., Wang, L. (eds.) MobiQuitous '20: Computing, Networking and Services, Virtual Event / Darmstadt, Germany, December 7–9, 2020, pp. 76–85. ACM (2020)
10. Meusel, R., Vigna, S., Lehmburg, O., Bizer, C.: Graph structure in the web—revisited: a trick of the heavy tail. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 427–432 (2014)
11. Steinmetz, K., Gerber, J.: “it doesn’t have to be this way”: Hacker perspectives on privacy. *Soc. Just.* **41**(3 (137), 29–51 (2015)
12. Liao, P., Zhao, H., Xu, K., Jaakkola, T.S., Gordon, G.J., Jegelka, S., Salakhutdinov, R.: Information obfuscation of graph neural networks. In: Meila, M., Zhang, T. (eds.) Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event. Proceedings of Machine Learning Research, vol. 139, pp. 6600–6610. PMLR (2021)
13. Li, K., Luo, G., Ye, Y., Li, W., Ji, S., Cai, Z.: Adversarial privacy-preserving graph embedding against inference attack. *IEEE Internet Things J.* **8**(8), 6904–6915 (2021)

14. Liu, R., Cao, Y., Chen, H., Guo, R., Yoshikawa, M.: FLAME: differentially private federated learning in the shuffle model. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021, pp. 8688–8696. AAAI Press (2021), <https://doi.org/10.1609/aaai.v35i10.17053>
15. Wang, D., Xu, J.: Principal component analysis in the local differential privacy model. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10–16, 2019, pp. 4795–4801. ijcai.org (2019)
16. Li, O., Sun, J., Yang, X., Gao, W., Zhang, H., Xie, J., Smith, V., Wang, C.: Label leakage and protection in two-party split learning. CoRR **abs/2102.08504** (2021)
17. Luo, X., Wu, Y., Xiao, X., Ooi, B.C.: Feature inference attack on model predictions in vertical federated learning. In: 2021 IEEE 37th International Conference on Data Engineering (ICDE), pp. 181–192. IEEE (2021)
18. Weng, H., Zhang, J., Xue, F., Wei, T., Ji, S., Zong, Z.: Privacy leakage of real-world vertical federated learning. arXiv preprint [arXiv:2011.09290](https://arxiv.org/abs/2011.09290) (2020)
19. Zhao, Y., Zhao, J., Yang, M., Wang, T., Wang, N., Lyu, L., Niyato, D., Lam, K.: Local differential privacy-based federated learning for internet of things. IEEE Internet Things J. **8**(11), 8836–8853 (2021)
20. Baek, C.H., Kim, S., Nam, D., Park, J.: Enhancing differential privacy for federated learning at scale. IEEE Access **9**, 148090–148103 (2021)
21. Kim, S.: Incentive design and differential privacy based federated learning: a mechanism design perspective. IEEE Access **8**, 187317–187325 (2020)
22. Yin, L., Feng, J., Xun, H., Sun, Z., Cheng, X.: A privacy-preserving federated learning for multiparty data sharing in social iots. IEEE Trans. Netw. Sci. Eng. **8**(3), 2706–2718 (2021)
23. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI Mag. **29**(3), 93–93 (2008)
24. Pan, L., Shi, C., Dokmanic, I.: Neural link prediction with walk pooling. CoRR **abs/2110.04375** (2021)
25. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint [arXiv:1609.02907](https://arxiv.org/abs/1609.02907) (2016)
26. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International Conference on Machine Learning, pp. 6861–6871. PMLR (2019)
27. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint [arXiv:1710.10903](https://arxiv.org/abs/1710.10903) (2017)
28. Ahn, S.J., Kim, M.: Variational graph normalized autoencoders. In: Proceedings of the 30th ACM International Conference on Information and Knowledge Management, pp. 2827–2831 (2021)
29. Kipf, T.N., Welling, M.: Variational graph auto-encoders. arXiv preprint [arXiv:1611.07308](https://arxiv.org/abs/1611.07308) (2016)
30. Melis, L., Song, C., Cristofaro, E.D., Shmatikov, V.: Exploiting unintended feature leakage in collaborative learning. In: 2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19–23, 2019, pp. 691–706. IEEE (2019)
31. Zhang, J., Zhang, Z., Xiao, X., Yang, Y., Winslett, M.: Functional mechanism: regression analysis under differential privacy. arXiv preprint [arXiv:1208.0219](https://arxiv.org/abs/1208.0219) (2012)
32. Fioretto, F., Van Hentenryck, P., Zhu, K.: Differential privacy of hierarchical census data: an optimization approach. Artif. Intell. **296**, 103475 (2021)
33. Chen, J., Lin, X., Xiong, H., Wu, Y., Zheng, H., Xuan, Q.: Smoothing adversarial training for GNN. IEEE Trans. Comput. Soc. Syst. **8**(3), 618–629 (2021)
34. He, X., Jia, J., Backes, M., Gong, N.Z., Zhang, Y.: Stealing links from graph neural networks. In: Bailey, M., Greenstadt, R. (eds.) 30th USENIX Security Symposium, USENIX Security 2021, August 11–13, 2021, pp. 2669–2686. USENIX Association (2021)
35. Chen, J., Wu, Y., Xu, X., Zheng, H., Ruan, Z., Xuan, Q.: PSO-ANE: adaptive network embedding with particle swarm optimization. IEEE Trans. Comput. Soc. Syst. **6**(4), 649–659 (2019)

36. Togami, M., Masuyama, Y., Komatsu, T., Nakagome, Y.: Unsupervised training for deep speech source separation with kullback-leibler divergence based probabilistic loss function. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 56–60. IEEE (2020)
37. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 308–318 (2016)
38. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint [arXiv:1412.6980](https://arxiv.org/abs/1412.6980) (2014)
39. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint [arXiv:1511.07289](https://arxiv.org/abs/1511.07289) (2015)

Chapter 16

Using Adversarial Examples to against Backdoor Attack in Federated Learning



16.1 Introduction

Federal learning (FL) has been applied to a wide range of different application domains because of its distributed training efficiency and privacy-preserving properties [1, 2]. In FL, each client executes training according to locally collected data and provides updated models to a centralized server. The server then summarizes these updated models into a new global model and also sends the new model back to each client. The training process is iterated until the global model converges. Based on the characteristics of the training data, FL can be broadly categorized into horizontal joint learning, vertical joint learning, and transfer joint learning [3].

Unfortunately, recent studies [4, 5] have found that horizontal federated learning is susceptible to backdoor updating of the model by a malicious client during the training phase, which is defined as a backdoor attack. It is not difficult to backdoor the global model since the client has full access and modification of the local training dataset. Many backdoor attacks [6–8] have been raised and have become a security threat hindering the development of FL. There has been extensive interest and effort to design robust aggregation algorithms to minimize the impact of backdoor attacks on FL.

Existing defenses are still challenged by either being bypassed by adaptive attacks or sacrificing FL's primary mission performance. The main goal of an effective defense against FLs is twofold: *(1) effective defense against both backdoor attacks and adaptive attacks; (2) maintaining the performance of FLs in their primary tasks after aggregation*. For the first goal, we plan to use anomaly detection on the server side to analyze the intrinsic difference between benign and backdoor models to make sure that malicious clients cannot bypass them. For the second goal, we only select benign updated models and try not to interfere with the aggregation process of those models to ensure FL's performance of the main task.

We present a novel FL backdoor defense method for short. In particular, we collect a small set of correctly labeled clean data as the detection dataset. For each client's update model, the server generates some adversarial examples using the

detection dataset. Our method is intuitively a time-consuming approach because sever generates adversarial examples for each client in each training cycle, so we design an intermittent strategy to trade-off detection effectiveness and efficiency. The server then tests all updated models by adversarial instances and clusters them using the activation values of the neurons in the penultimate layer of each model to accept those updated models that are classified as “benign”. Backdoor detection is a model of server anomaly detection problem based on neuron activation values under adversarial attack. Because clustering distance plays an essential part in anomaly detection. In our method, the clustering results are not only used for backdoor model detection, but also for updating the trust score to reflect the credibility of the client. Therefore, our method applies weighted aggregation based on each client’s trust score, so that even if a backdoor update model is inadvertently received, the backdoor attack will fail due to a low trust score.

Our main contributions to the work can be summarized as follows:

- (1) By conducting adversarial attacks on the benign model and backdoor model, we gain an insight that adversarial perturbations can activate backdoor neurons in the backdoor model.
- (2) Motivated by this, we present a novel FL backdoor defense method using adversarial instances to defend against backdoor attacks in FL. Our method performs backdoor detection at breakpoints by clustering the neuron activation values of each updated model based on the neuron activation values fed from the adversarial examples generated intermittently from the detection dataset.
- (3) We conduct extensive experiments with four models on four datasets and show that our method achieves state-of-the-art (SOTA) effectiveness in defending against backdoor attacks in FL. We also show that our method is robust to adaptive attacks.

16.2 Background and Related Work

Many secure aggregation algorithms have been proposed to mitigate or even eliminate the effects of backdoor attacks before aggregation. These studies propose that the update models of malicious clients have unique characteristics. They can identify malicious updates and reduce their updates by comparing the similarity of each client [9–12]. Other works tried to replace the average value with robust estimation, such as coordinate median, geometric median α -Trim average, or combination of these techniques [10, 13–15]. Ozdayi *et al.* [16] even further attempted to improve the robustness of FL by assigning different learning rates to each client. Furthermore, recent work focused on eliminating backdoors in updated models. Sun *et al.* [7] demonstrated that tuning and updating the model’s specification with the addition of Gaussian noise can be effective in mitigating backdoor attacks. Xie *et al.* [17] utilized cropping and smoothing of model parameters in order to contain the global model’s smoothness, resulting in a proof-of-instance robustness to a finite number of

backdoors. However, the robustness of these approaches is worth exploring given the different data distributions of clients in FL and the possibility that malicious clients may limit updating the model. Some work in parallel [18, 19] in each FL round adds an additional validation phase to detect backdoors. These methods emphasize on client-side verification, which requires multiple communications between the client and the server.

16.3 Threat Model

16.3.1 Attacker's Goal

During the training phase, malicious clients intentionally scramble the training data (e.g., using triggers) to misrepresent the model. Upon completion of training, any test example with a particular trigger is incorrectly categorized with the target label [20]. The backdoor client in FL manipulates the local model through simultaneously fitting the main task and the backdoor attack. The backdoor data, in other words, is used to train the model for the backdoor task, in contrast to the clean data, which ensures good performance on the main task. As a result, the global model works properly with the clean example without triggers, while a high attack success rate is achieved on the backdoor example with triggers.

Suppose that the target of a vicious client model w'_i in an ephemeral element t with local dataset D_i and target label τ can be represented as

$$\begin{aligned} w'_i = \arg \max_{w_i} & \left(\sum_{j \in D_i^{cln}} P[w_g^t(x_j) = y_j] \right. \\ & \left. + \sum_{j \in D_i^{bak}} P[w_g^t(\Theta(x_j, \phi)) = \tau] \right), \end{aligned} \quad (16.1)$$

where w_g^t is the global model, P represents the model output probability, D_i^{cln} is the clean data, and D_i^{bak} is the backdoor data of client i . Here, D_i^{cln} and D_i^{bak} satisfy $D_i^{bak} \cap D_i^{cln} = \emptyset$ and $D_i^{bak} \cup D_i^{cln} = D_i$. For CBA, ϕ is the only one unified trigger, for DBA, $\phi = \{\phi_1, \dots, \phi_n\}$ is a set of variance triggers. The function Θ converts clean data in any tag to backdoor data with trigger ϕ . Since CBA and DBA are two typical and productive backdoor attacks in FL, we use them as benchmarks to assess our defenses.

16.3.2 Attacker's Capability

This paper assumes that the server is honest, yet at least one client is malignant. If there are multiple clients, it is possible for them to communicate and collude with each other and manipulate their own training data, but they cannot visit or manipulate

the data of the benign client or its learning process. In addition, a malicious client cannot manipulate the server’s loss function computation, optimization algorithm, or aggregation process. We assume for more general applications that the clients’ data are non-independent and co-distributed, which certainly has more realistic implications and makes defense more difficult.

16.3.3 *Defender’s Knowledge and Capability*

Designing a defense method to detect whether each locally updated model is a backdoor model or a benign model for FL, with the server playing the role of the defender. More practically, suppose the server does not have access to the client’s raw local training data and does not know the number of malicious clients. Nevertheless, the server has complete visibility to the global model and to the locally updated models of all clients in each calendar element. Furthermore, the server itself can request the labels used for training from each client in order to collect a clean small training dataset (defined as the detection dataset) to be used for the detection task. It is useful and reasonable in practical applications and was proposed in Cao’s work [21]. For example, Google asks its employees to create a detection dataset [21] using gboard for co-predicting the next word. Because only a small number of detection datasets are used for defense, the server can take on the task of manually collecting and labeling these datasets.

16.4 Methodology

This section describes the suggested method of defense. In the preparation phase of FL, the server maintains the appropriate detection dataset itself. After beginning the training of FL, our method performs two steps, model testing and model aggregation. As illustrated in Fig. 16.1, specifically, in the first step, the server receives the updated model from the client and tests the updated model using the generated adversarial paradigm. The server will update the adversarial examples according to the global model training loss. Once all update models are tested, the server utilizes a K-means clustering algorithm [22] to categorize the update models as “benign”, “malicious”, “suspected benign”, and “suspected malicious”. In the second step, the server receives “benign” updates to the model and summarizes them according to the client’s trust score. Finally, the server updates the client’s trust scores based on the results of the tests during this period.

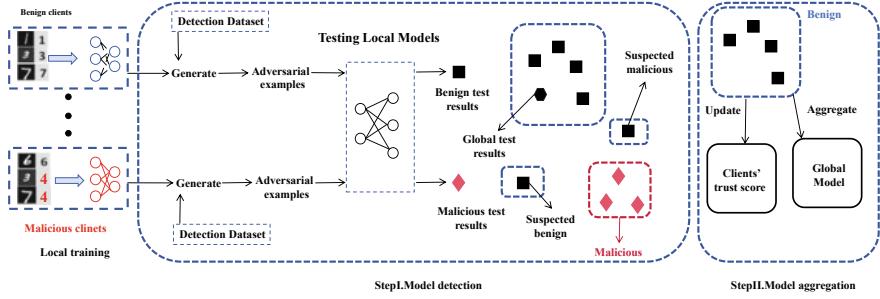


Fig. 16.1 FL training overview of our method. This process includes two steps: model detection and model aggregation

16.4.1 Model Detection

Motivated by the fact that adversarial examples are more likely to trigger backdoors in the model than benign examples, we employ a small dataset \mathcal{R}^d consisting of clean examples and correct labels [21] on the server side to generate adversarial examples for each client. Server then tests each local update model using adversarial instances and the results of each client's test are compared to confirm whether the client is benign or has a backdoor. As PGD is broadly used as an attack for verifying model robustness [23], and it may be considered as the strongest first-order attack [24, 25], it is a suitable choice for our method as well. Taking into account the detection dataset distribution $(x, y) \in \mathcal{R}^d$, the appropriate loss function can be represented as $\mathcal{L}(\theta, x, y)$, with θ being the set of parameters. PGD is used by the server on the negative loss function to obtain the counter example x_{adv} :

$$x_{adv}^{t+1} = \Pi_{x+\mathcal{S}} (x_{adv}^t + \alpha sign (\nabla_x \mathcal{L}(\theta, x, y))), \quad x_{adv}^0 = x, \quad (16.2)$$

where t is the current number of iterations, x^t is the confrontation example after iteration t , $\Pi_{x+\mathcal{S}}$ restricts the confrontation perturbation to a spherical range. ∇_x denotes the gradient, $sign()$ picks up the orientation of the gradient, α denotes the amplitude of the image's pixel update per iteration.

We choose the global model as the benchmark for judgment. Particularly, a local update model may be more “promising” if the current test results of the local update model are more similar to the test results of the global model. Since the number of clusters is known in advance, a simple and effective method is to group the local update models using K-means. Based on our insights, neurons that play a backdoor role in the model are triggered by adversarial perturbations. These neurons tend to have abnormally large activation values, which leads to abnormal outputs in the penultimate layer of the model [26, 27].

Since malicious clients have the same backdoor target, the neurons of their model's penultimate layer will have anomalous activation values when fed with adversarial examples. Thus, the test results can serve as a metric for delineating the updated

model. The test result \mathcal{X}_i for the L-1 layer, given a model w_i with an L-layer network, under the adversarial example (x_{adv}, y) is

$$\mathcal{X}_i = \sum_{j=1}^{\mathcal{R}^{d'}} f_i^{L-1}(x_{adv}^j), \quad (16.3)$$

where $\mathcal{R}^{d'}$ is the set of adversarial examples, f_i^{L-1} is the output from the $(L - 1)_{th}$ layer of the model.

Once all the updated models have been tested, the server can obtain a set of test results $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$. Then, according to the current test results of the global model \mathcal{X}_g , server can utilize K-means clustering to classify the test results into four categories: “benign”, “malicious”, “suspected benign” and “suspected malicious”. It is difficult to explicitly classify all clients as “benign” and “malicious” given the heterogeneity of client data in FL. Therefore, the server simply accepts “benign” update models and excludes “malicious”, “suspected benign”, and “suspected malicious” update models. The value of K in the K-means algorithm can be identified as 4. Suppose the server first randomly chooses 4 test results $\{C_1, C_2, C_3, C_4\}$ in the test result $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$. The Euclidean distance from each result \mathcal{X} to each clustering center can then be calculated as

$$\sum_{i=1}^n \sum_{k=1}^4 dis(\mathcal{X}_i, C_k) = \sqrt{(\mathcal{X}_i - C_k)^2}. \quad (16.4)$$

Based on the Euclidean distance, each \mathcal{X} will be assigned to the nearest cluster. After all test results $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$ are assigned, based on the average of all test result distances $\{\mathcal{X}_1, \mathcal{X}_2, \dots, \mathcal{X}_n\}$ distance to the cluster center in a cluster, then iteratively assigns \mathcal{X} and updates the cluster center for multi-epoch iteration. Once clustered, the server needs to compute the Euclidean distance of each cluster center to the global test result \mathcal{X}_g , and defined as “benign” (denoted as Set_b) the set of updated models that are closest to \mathcal{X}_g in a cluster.

16.4.2 Model Aggregation

After K-means clustering divides the updated models, the server aggregates the updated models in Set_b to get a new global model. It should be noted that after experiments, we found that the clustering method does not 100. Therefore, we propose to assign a trust score to each client during the training process. We set the trust score for n clients as $T = \{T_1, T_2, \dots, T_{n-1}, T_n\}$. The trust score T_i of the i_{th} client in the new epoch t is as follows:

$$\mathcal{T}_{i,t} = \begin{cases} \mathcal{T}_{i,t} + 1 & \mathcal{X}_i \in Set_b, t \geq 1 \\ \mathcal{T}_{i,t} & \mathcal{X}_i \notin Set_b, t \geq 1 \\ 1 & t = 0 \end{cases}. \quad (16.5)$$

Trust scores are accumulated from the number of times the server accepts each client. Each client has an initial trust score of 1. In each epoch, once a client's updated model is labeled as "benign", the client's trust score increases to 1. In short, the higher the client's trust score, the more frequently the server accepts that client, and the more "trustworthy" the client's updated model is.

Thus, the final global model aggregation equation can be represented as follows:

$$w_g = \sum_{i=1}^n \frac{\mathcal{T}_i}{\mathcal{T}} w_i, \quad (16.6)$$

where $\mathcal{T} = \sum_{i=1}^n \mathcal{T}_i$. During the aggregation process, the server selects only "benign" update models for weighted aggregation based on the client's trust score. This way, if some models with malicious updates are incorrectly selected at a certain stage, they are assigned a lower weight due to a lower trust score, thus not affecting the global model. Our method's only difference from the FedAvg algorithm is that our method excludes a certain number of malicious clients during the model aggregation step. According to the work [16], our method has the same convergence as FedAvg algorithm [28].

Furthermore, we note that generating confrontation examples for updated models at each epoch increases the complexity of our method. Therefore, it is important to think about updating the adversarial examples at intervals. We adopt a hyperparameter for our method called the update interval γ . In particular, after summarizing the new global model w_g , our method will use the detection dataset \mathcal{R}^d to compute the loss of the global model \mathcal{L} .

The server can get an updated loss value $\Delta\mathcal{L}$ by comparing it to the loss of the previous epoch. The updated loss value of $\Delta\mathcal{L} \geq \gamma$ indicates that the global model has been significantly updated, and the adversarial examples require to be updated prior to the next test to make sure that the adversarial examples can accommodate the latest update of the model. Otherwise, if the updated loss value $\Delta\mathcal{L} < \gamma$, our method does not need to update adversarial examples. This guarantees that the adversarial examples are able to adapt to the updated model in a timely manner without the need to regenerate the adversarial examples in each time zone.

16.5 Experiments

A malicious client in our experiments performs a backdoor attack during the training process of FL, while the server endeavors to make sure that the global model is not vulnerable to the backdoor attack. In particular, we assess the protection effect of

our method and interpret its effectiveness intuitively. We then test our method in an adaptive scene and conduct a parameter susceptibility analysis.

16.5.1 Datasets and Models

We use four datasets for FL training: MNIST [29], Fashion-MNIST [30], CIFAR-10 [31], and LFW [32]. MNIST, Fashion-MNIST, CIFAR-10 are three general image classification datasets, and all three datasets are 10 classification tasks. LFW is an unrestricted natural scene face identification dataset, which consists of more than 13,000 face images of celebrities around the world. We selected the top 50 characters in the 50 classification tasks and expanded the data by rotational transformation. In addition, we verified the portability of our method using the CIFAR-100 [31] dataset. For all datasets, 80% of the data is used for training and 20% for testing. M_p is the proportion of malicious clients and l_r is the learning rate.

16.5.2 Experiment Setup

Training Settings. We focus on realistic FL setups in which data is distributed among clients in non-i.i.d [33] manner. We distribute the training data among clients using a Dirichlet distribution with a distribution hyperparameter of $\alpha = 0.5$ and use client uses SGD as the optimizer. Local training is done 5 epochs for each benign client and 10 epochs for malicious clients. We choose 20 images from the test dataset for each label to compose the detection dataset to ensure that the test model has never seen the test data. Regarding the LFW dataset, we choose 5 images constituting the detection dataset for each label. On the server side, we tested the performance of two secure aggregation algorithms, i.e., Krum [13], FoolsGold [12] with. Unless otherwise specified, for our method, we set the clustering number $c = 4$, the update interval $\gamma = 0.2$, and the perturbations of PGD $\epsilon = 0.4$.

Attack Settings. We assume 40% of clients are malicious, and they can collude with each other. We choose CBA and DBA as the attack methods, employing the backdoor task of setting triggers that cause the model to misclassify instances of the base label as target labels. For the datasets MNIST, Fashion-MNIST, and CIFAR-10, we assign white color to the selected pixel points, and the selected pixel points are the four pixel points in the lower right corner of the image. The goal of the attack is to categorize “6” as “9”. For dataset LFW, we assign white color to selected pixels, and the selected pixels are the 16 pixels in the bottom right corner of the image. The goal of the attack is to categorize all labels as “9”. To implement the DBA, we categorize the triggers described above into four and deploy them on different clients.

Metrics. Assume that the backdoor example labeled as target label is *TN* (*True Negative*) and the backdoor example not labeled as target label is *FN* (*False Negative*).

The clean example for predictive categorization is then *TP* (*True Positive*) and the clean example for unpredictive categorization is *FP* (*False Positive*). When the FL backdoor attack is valid, we use the attack success rate as a metric to verify the validity of the attack. The *backdoor attack success rate* is defined as follows:

$$ASR = TN / (TN + FN). \quad (16.7)$$

The *classification accuracy* is defined as follows:

$$ACC = TP / (TP + FP). \quad (16.8)$$

We use *defense efficiency rate* (DER) to gauge the effectiveness of the security aggregation algorithm. It reflects the impact of security aggregation algorithms on ACC and ASR. DER is defined as follows:

$$DER = ACC * (1 - ASR). \quad (16.9)$$

Platform. CPU is Intel XEON 6240 2.6GHz x 18C, GPU is Tesla V100 32GiB, the Memory is DDR4-RECC 2666 16GiB, the operating system is Ubuntu 16.04, the programming language is Python 3.6, and the deep learning framework is Pytorch-1.4.0.

16.5.3 Effectiveness of Our Method

Using the same attack setup, we evaluate the defense performance of our method against CBA and DBA attacks. The results are shown in Tables 16.1 and 16.2. Our method attains SOTA performance under both CBA and DBA attacks. It makes sure that the ASR is minimized without affecting the ACC. On the MNIST and Fashion-MNIST datasets, our method does not have the best results at the beginning, but as the attack continues, the FoolsGold and Krum methods do not decrease their DER values and they defend well. However, their DER values are lower than our method's, which indicates that our method's effect on ACC is less than these two methods. Moreover, our method has better performance on the CIFAR-10 and LFW datasets, and its DER value is much higher than the other two methods.

16.5.4 Interpretability of Our Method via Neuron Activation

Adversarial perturbations trigger the activation of backdoor neurons in the model, resulting in unusual output from neurons in the penultimate layer of the model. We visualize the neurons in the penultimate layer of the backdoor model activated on

Table 16.1 Defensive effect under CBA attack (DER value)

Dataset	Defend methods	Epoch0	Epoch10	Epoch20	Epoch30	Epoch40
MNIST	FoolsGold	25	148	150	165	170
	Krum	25	175	185	189	190
	Our method	28	185	195	198	200
	FoolsGold	125	155	160	165	165
Fashion-MNIST	Krum	100	132	148	150	154
	Our method	115	160	164	167	170
	FoolsGold	18	5	19	35	39
CIFAR-10	Krum	15	20	39	43	48
	Our method	19	20	40	59	78
	FoolsGold	25	155	119	0	0
LFW	Krum	0	175	168	175	175
	Our method	45	199	200	200	200

Table 16.2 Defensive effect under DBA attack (DER value)

Dataset	Defend methods	Epoch0	Epoch10	Epoch20	Epoch30	Epoch40
MNIST	FoolsGold	30	160	175	183	192
	Krum	50	170	175	176	178
	Our method	50	183	192	195	198
	FoolsGold	125	149	153	160	165
Fashion-MNIST	Krum	49	125	139	143	148
	Our method	125	166	175	177	180
	FoolsGold	20	18	0	8	10
CIFAR-10	Krum	5	4	19	37	48
	Our method	24	26	38	50	68
	FoolsGold	20	100	189	0	0
LFW	Krum	74	153	0	0	0
	Our method	78	200	201	200	200

the MNIST dataset, including under the clean example, adversarial example, and backdoor example inputs, as shown in Fig. 16.2.

It can be seen obviously that the activation values of the neurons of the model when fed with clean examples are very different from the activation values when fed with backdoor examples. This demonstrates that there are really some backdoor neurons responsible for the backdoor mission in the backdoor model. These neurons are only activated when the backdoor triggers are input. When the adversarial

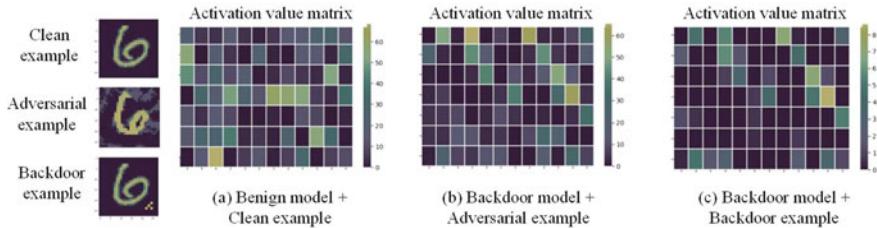


Fig. 16.2 The neuron activation values of the benign model and the backdoor model under different examples, the adversarial perturbations can trigger the backdoor neurons in the backdoor model

Table 16.3 The IoU of neuron activation of the backdoor model under different example inputs

	MNIST	Fashion-MNIST	CIFAR-10	LFW
Adversarial examples	82	65	80	86
Clean examples	30	40	42	34

examples are input, the activation values of the neurons are very similar to the input backdoor example as can be observed from Fig. 16.2b. It suggests that adversarial perturbations can mobilize the backdoor neurons in the backdoor model, leading to misclassification of the model and obtaining a higher attack success rate than the benign model.

Furthermore, we compute the Intersection over Union (IoU) [34, 35] of neuron activation under different example inputs. In particular, we enumerate and sort the activations of neurons in the penultimate layer of the model. IoU means the similarity of the neurons in the model to the activations of two different inputs. The higher the IoU is, the more similar the neurons’ activations are.

We compare the neurons’ activation IoU of the backdoor model neurons under different inputs, as shown in Table 16.3. The “Adversarial Example” is the ratio of neuron activation IoU when inputting the adversarial examples and the backdoor examples, and the “Clean Example” is the ratio of neuron activation IoU when inputting the Clean Example and the Backdoor Example. As can be seen, the neuron activation values when inputting the adversarial paradigm are more comparable to those when inputting the backdoor paradigm, which indicates that the backdoor neurons in the model are successfully activated by the adversarial paradigm. Therefore, the backdoor model has a higher confrontation success rate than the benign model, so we can use confrontation instances to distinguish the benign model from the backdoor model.

Table 16.4 The defense effect of our method under the adaptive attack

	MNIST	Fashion-MNIST	CIFAR-10	LFW
ACC-Boundary	98.33%	86.83%	75.12%	99.68%
ASR-Boundary	12.18%	2.19%	1.96%	2.43%
ACC-PGD	97.26%	87.26%	77.29%	99.79%
ASR-PGD	4.79%	6.37%	1.84%	2.35%

16.5.5 Our Method Against Adaptive Attacks

In the real world, the attacker is likely to have mastered the details of the FL's defense and performed an adaptive attack. Therefore, it is essential to study the defense capability of our method against adaptive attacks. The categorization basis of our approach is that backdoor models will be more fragile than benign models under adversarial examples. Supposing malicious clients have a good prior knowledge of our method, they can evade detection by enhancing the robustness of the backdoor model to adversarial examples. In particular, malicious clients can add some adversarial examples for adaptive training during the training process to make sure the backdoor model is robust to adversarial examples. We suppose that there are two scenarios: (1) the malicious clients know the server's method for generating adversarial examples, and they use the same method (PGD attack) to generate the examples; (2) the malicious clients do not know the server's method for generating adversarial examples, and they use a different method (boundary attack [36]) to generate examples.

We use the same device as in 5.3 and select CBA as the backdoor method. Suppose the malicious client replaces 30% of the training data randomly with adversarial examples. Table 16.4 illustrates the results of our experiments. By comparison, we can see that our method still has a defense effect regardless of whether the malicious client knows our detection method or not. Our method makes sure that the global model will not be implanted with a backdoor, which indicates that the backdoor model is still different from the benign model even if the malicious client is trained adaptively. In order to explain the reason why our method can successfully defend against adaptive attacks, we examine the test results of the benign and backdoor models after adversarial training. Figure 16.3 presents the results, where *Backdoor(Adv)* denotes the adversary-trained backdoor model. We notice that the adversary-trained backdoor model is robust to adversarial examples. The clean example of its neuron activation is very similar to the adversarial example. Nevertheless, since a malicious client has no way of knowing the robustness of other benign client models, this tends to cause the backdoor model to exhibit anomalous robustness, which is also different from the benign models (as shown in Fig. 16.3b). As a result, the backdoor model is still easily recognized by our method even if the malicious client performs an adaptive attack.

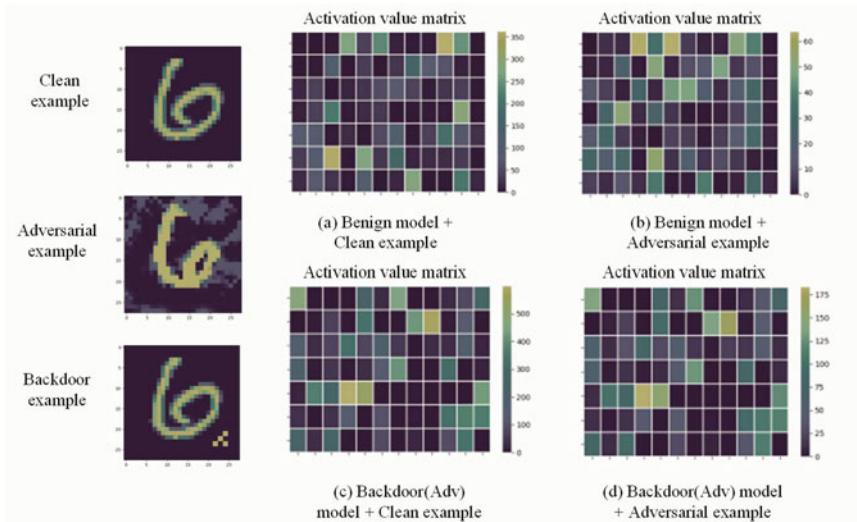


Fig. 16.3 The test results for the benign model and the backdoor model after adversarial training

16.6 Conclusion

In this work, we present a new FL defense approach to defend against backdoor attacks in FL. The key to our method is that the server collects detection datasets and then produces adversarial examples to test local update models. Our method then uses a clustering algorithm to choose benign update models for clustering. Our extensive experiments on four image datasets show that our method demonstrates SOTA defense performance. Moreover, our method remains robust under adaptive attacks.

References

1. Li, L., Fan, Y., Tse, M., Lin, K.: A review of applications in federated learning. *Comput. Ind. Eng.* p. 106854 (2020)
2. Li T, Sahu AK, Talwalkar A, Smith V (2020) Federated learning: Challenges, methods, and future directions. *IEEE Signal Process. Mag.* 37(3):50–60
3. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.* **10**(2), 12:1–12:19 (2019)
4. Mothukuri, V., Parizi, R.M., Pouriyeh, S., Huang, Y., Dehghantanha, A., Srivastava, G.: A survey on security and privacy of federated learning. *Future Gener. Comput. Syst.* pp. 619–640 (2021)
5. Ma C, Li J, Ding M, Yang HH, Shu F, Quek TQS, Poor HV (2020) On safeguarding privacy and security in the framework of federated learning. *IEEE Netw.* 34(4):242–248

6. Wang, H., Sreenivasan, K., Rajput, S., Vishwakarma, H., Agarwal, S., Sohn, J., Lee, K., Papailiopoulos, D.S.: Attack of the tails: Yes, you really can backdoor federated learning. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020)
7. Sun, Z., Kairouz, P., Suresh, A.T., McMahan, H.B.: Can you really backdoor federated learning? CoRR (2019)
8. Bagdasaryan, E., Veit, A., Hua, Y., Estrin, D., Shmatikov, V.: How to backdoor federated learning. In: Chiappa, S., Calandra, R. (eds.) The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]. pp. 2938–2948. Proceedings of Machine Learning Research (2020)
9. Chen, Y., Su, L., Xu, J.: Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. In: Psounis, K., Akella, A., Wierman, A. (eds.) Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 2018, Irvine, CA, USA, June 18-22, 2018. p. 96 (2018)
10. Mhamdi, E.M.E., Guerraoui, R., Rouault, S.: The hidden vulnerability of distributed learning in byzantium. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. pp. 3518–3527. Proceedings of Machine Learning Research (2018)
11. Fu, S., Xie, C., Li, B., Chen, Q.: Attack-resistant federated learning with residual-based reweighting. CoRR (2019)
12. Fung, C., Yoon, C.J.M., Beschastnikh, I.: The limitations of federated learning in sybil settings. In: Egele, M., Bilge, L. (eds.) 23rd International Symposium on Research in Attacks, Intrusions and Defenses, RAID 2020, San Sebastian, Spain, October 14-15, 2020. pp. 301–316 (2020)
13. Blanchard, P., Mhamdi, E.M.E., Guerraoui, R., Stainer, J.: Machine learning with adversaries: Byzantine tolerant gradient descent. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 119–129 (2017)
14. Pillutla, V.K., Kakade, S.M., Harchaoui, Z.: Robust aggregation for federated learning. CoRR (2019)
15. Yin, D., Chen, Y., Ramchandran, K., Bartlett, P.L.: Byzantine-robust distributed learning: Towards optimal statistical rates. In: Dy, J.G., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018. pp. 5636–5645. Proceedings of Machine Learning Research (2018)
16. Özdayı, M.S., Kantarcıoglu, M., Gel, Y.R.: Defending against backdoors in federated learning with robust learning rate. In: Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021. pp. 9268–9276 (2021)
17. Xie, C., Chen, M., Chen, P., Li, B.: CRFL: certifiably robust federated learning against backdoor attacks. CoRR (2021)
18. Andreina, S., Marson, G.A., Möllering, H., Karame, G.: Baffle: Backdoor detection via feedback-based federated learning. In: 41st IEEE International Conference on Distributed Computing Systems, ICDCS 2021, Washington DC, USA, July 7-10, 2021. pp. 852–863 (2021)
19. Zhao L, Hu S, Wang Q, Jiang J, Shen C, Luo X, Hu P (2021) Shielding collaborative learning: Mitigating poisoning attacks through client-side detection. IEEE Trans. Dependable Secur. Comput. 18(5):2029–2041
20. Gu, T., Liu, K., Dolan-Gavitt, B., Garg, S.: Badnets: Evaluating backdooring attacks on deep neural networks. IEEE Access pp. 47230–47244 (2019)
21. Cao, X., Fang, M., Liu, J., Gong, N.Z.: Fltrust: Byzantine-robust federated learning via trust bootstrapping. In: 28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021 (2021)

22. Tzortzis G, Likas A (2009) The global kernel k -means algorithm for clustering in feature space. *IEEE Trans. Neural Networks* 20(7):1181–1194
23. Frey, B.J., Dueck, D.: Clustering by passing messages between data points. *science* **315**(5814), 972–976 (2007)
24. Chang, T., He, Y., Li, P.: Efficient two-step adversarial defense for deep neural networks. *CoRR* (2018)
25. Tramèr, F., Boneh, D.: Adversarial training and robustness for multiple perturbations. In: Wallach, H.M., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E.B., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada.* pp. 5858–5868 (2019)
26. Liu, Y., Lee, W., Tao, G., Ma, S., Aafer, Y., Zhang, X.: ABS: scanning neural networks for back-doors by artificial brain stimulation. In: Cavallaro, L., Kinder, J., Wang, X., Katz, J. (eds.) *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019.* pp. 1265–1282 (2019)
27. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I.M., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. In: Espinoza, H., hÉigeartaigh, S.Ó., Huang, X., Hernández-Orallo, J., Castillo-Effen, M. (eds.) *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence 2019 (AAAI-19), Honolulu, Hawaii, January 27, 2019. CEUR Workshop Proceedings* (2019)
28. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Singh, A., Zhu, X.J. (eds.) *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA.* pp. 1273–1282. *Proceedings of Machine Learning Research* (2017)
29. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proc. IEEE* 86(11):2278–2324
30. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR* (2017)
31. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. *Tech. Rep.* 0, University of Toronto, Toronto, Ontario (2009)
32. Huang, G.B., Mattar, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In: *Workshop on faces in’Real-Life’Images: detection, alignment, and recognition* (2008)
33. Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., Chandra, V.: Federated learning with non-iid data. *CoRR* (2018)
34. Li, Y., Li, Y., Lv, Y., Jiang, Y., Xia, S.: Hidden backdoor attack against semantic segmentation models. *CoRR* (2021)
35. Lin, J., Xu, L., Liu, Y., Zhang, X.: Composite backdoor attack for deep neural network by mixing existing benign features. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) *CCS ’20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020.* pp. 113–131 (2020)
36. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings* (2018)

Part III

Testing for Deep Learning

Testing and evaluation of deep neural networks are effective methods for measuring the security and robustness of deep learning models. Testing and evaluation can be divided into constructing an indicator system and testing methodology. However, there are several main issues with current testing and evaluation methods:

1. Currently, evaluating the robustness of deep models is time-consuming and requires specific attacks and model structures.
2. The backdoor detection methods currently existing have limitations in terms of sensitivity to trigger size and high requirements for training data.
3. Lack of proof, effectiveness, and generalizability.
4. The effectiveness of generation is affected by gradient disappearance, making it difficult to extend to unstructured data.

Chapter 17 introduces a robustness evaluation metric based on the feature distribution of models that is attack-independent, solving the problem of current evaluation methods being time-consuming and requiring specific attacks and model structures. This method includes two aspects: feature subspace aggregation within the same class and feature subspace distance between different classes. FSA describes the compactness of data within the same class, while FSD describes the difference between different classes. This method integrates FSA and FSD into one metric, with a smaller calculated value indicating stronger robustness of the deep model.

Chapter 18 introduces a test input prioritization technique designed from the perspective of the moving cost of test inputs in the DNN feature space, addressing the lack of proof, effectiveness, and generalizability of current testing methods. This method simplifies the problem of measuring improper behavior probability to measuring the difficulty of movement in the feature space. Based on this, a formal guarantee of the lower bound of moving cost is provided, and the moving cost value is calculated using GEVT to complete the testing evaluation.

Chapter 19 introduces a testing method based on AS curves and AUC measurements, addressing the issue that evaluation effectiveness is affected by gradient disappearance. This method qualitatively and quantitatively explains the severity

of discrimination at each layer of DNN, systematically searches the input space, and performs testing evaluations on the model.

Chapter 20 discusses the application of testing methods in real life, introducing an end-to-end dynamic link prediction for deep learning models. This method can automatically learn the structural and temporal features of networks within a unified framework and predict links that have never appeared in the network.

This book provides a comprehensive overview of the security challenges faced by deep learning models in various fields and proposes practical solutions to mitigate these challenges. By considering different data modalities, model architectures, and task objectives, it aims to uncover the diverse security issues that arise in deep learning systems and propose practical solutions to alleviate them. This book is suitable for researchers, practitioners, and students interested in the security aspects of deep learning, and aims to provide valuable resources for further research and development in this rapidly evolving field.

Chapter 17

Evaluating the Adversarial Robustness of Deep Model by Decision Boundaries



17.1 Introduction

In this chapter, our focus lies in evaluating the adversarial robustness of deep models, which has become a critical concern directly impacting their practical applications. However, existing methods for evaluating robustness often rely on different attack results and calculate the attack success rate (ASR) as a measure of robustness [1]. This approach leads to the entanglement of model robustness with the specific attack algorithms used, resulting in high computational costs and limitations in robustness analysis due to the capabilities of the attacks themselves. Moreover, the dependency between robustness evaluation and attack approaches can introduce biases in the analysis. To address these challenges, various certified robustness evaluation methods have been proposed, involving model certifiers and metrics [1, 2]. These methods typically analyze specific layers and activation functions of the network to prove the model's robustness, requiring detailed information about the model's structure. However, these certified methods often come with high computational costs, and their white-box nature limits their practicality.

The significance of decision boundaries in classification has been widely acknowledged [3]. A well-trained classifier assigns all points on one side of the decision boundary to a specific class, while the remaining points on the other side are assigned to a different class. The decision boundary is intuitively linked to the classification performance of the classifier. This understanding has led to numerous research efforts focusing on robustness. He et al. [4] investigated the properties of decision boundaries to assess the effectiveness of adversarial attacks, albeit limited to specific attack scenarios. Jiang et al. [5] and Karimi et al. [6] measured decision boundaries based on adversarial samples but did not achieve attack-agnostic measurements. Singla et al. [7] calculated the distance from samples to the decision boundary of a deep ReLU network, yet the measurement of ReLU-free networks remained unexplored. DeepXplore [8] generated test samples near the decision boundary by maximizing neuron coverage, but whether the level of neuron coverage directly correlates with model robustness requires further examination.

Extensive research has demonstrated that samples vulnerable to attacks tend to be positioned closer to decision boundaries [9–11]. Recent studies have also shown that decision boundaries progressively shift closer to natural samples during training, while adversarial training methods may potentially impede this convergence [12]. Moreover, the relationship between decision boundaries and the distribution of samples can be examined in a more nuanced manner. Formally, this examination involves assessing the intra-class and inter-class distances of vectors separated by the decision boundary. The intra-class distance quantifies the clustering degree within a class, while the inter-class distance measures the separation between the centers of two clusters. By minimizing intra-class distances and maximizing inter-class distances, the decision boundary becomes more distinct, ultimately contributing to a more robust model.

Drawing inspiration from the aforementioned concepts of decision boundary and sample distribution distance, we propose a novel adversarial robustness evaluation metric called ROBY (robustness evaluation based on boundary and feature space). ROBY combines the evaluation of the model’s decision boundary and feature space to assess its robustness. By quantitatively capturing the decision boundary of a deep classifier through the distribution of data around it, ROBY provides a robustness evaluation without requiring additional information about the model’s architecture and network composition, similar to certified robustness evaluation methods. Furthermore, compared to attack-based robustness evaluation methods, ROBY offers improved computational efficiency. One significant advantage of ROBY is its independence from adversarial samples, sample generation algorithms, and attack algorithms. Once a model is trained, it can compute its ROBY metric, making it a versatile and attack-agnostic evaluation approach. Our experiments demonstrate that ROBY achieves comparable performance to the Attack Success Rate (ASR) in evaluating the adversarial robustness of many state-of-the-art deep models. These models include DenseNet [13], MobileNetV1 [14], MobileNetV2 [15], ResNet50 [16], ResNet101 [16], InceptionV1 [17], InceptionV2 [18], AlexNet [19], SqueezeNet [20], LeNet [21](LeNet-5), and other deep models with varying complexities and capacities.

In conclusion, this chapter presents the following key contributions:

- We proposed a generic model robustness evaluation metric based on the decision boundaries. It is efficient, lightweight, attack-independent, and adversarial sample free.
- Our method achieves a strong positive correlation with attack success rate, the most adopted robustness evaluation metric.
- We applied our method to ten state-of-the-art neural network models, models with different complexity, and networks defended against projected gradient descent (PGD) [22] adversarial samples to verify its practicability and effectiveness. Our study is the first light-weighted robustness metric that is generic across deep models to the best of our knowledge.

17.2 Related Works

17.2.1 Robustness Evaluation of Deep Models

The conventional approach for evaluating robustness involves measuring the upper bound of the minimum adversarial distance using adversarial samples generated by attack algorithms [23, 24]. In general, if it is easier to construct an adversarial sample on a deep model with minimal perturbations, it indicates a lower level of robustness [25]. The attack success rate serves as a metric to assess the robustness of a model, with a more robust model having a lower attack success rate. This attack-based evaluation is typically conducted using state-of-the-art attack algorithms [22, 26]. However, this type of robustness evaluation method is still computationally intensive and can lead to biased evaluation conclusions.

Apart from assessing robustness through adversarial attacks, the robustness of a deep model can also be evaluated by considering its structure and parameters. Wang et al. [27] utilized topological terms to investigate the robustness of deep models; however, they did not provide robustness boundaries or estimations specifically for neural networks. In contrast, Gopinath et al. [28] proposed an automatic, data-driven approach called Depnaga. This method clusters labeled data and leverages existing constraint solvers to automatically identify the robust region of the network, where all inputs are accurately classified.

Certified robustness evaluation methods offer a provable lower bound on the minimum adversarial distance required to cause misclassification, typically achieved through well-designed metrics or model certifiers. For instance, the research conducted by Szegedy et al. [29] involved calculating the global Lipschitz constant for each layer of a neural network to explain the model's robustness. However, the global Lipschitz constant often provides a loose bound. Hein et al. [30], on the other hand, established a lower bound on robustness by utilizing local Lipschitz continuity conditions. They derived a closed-form bound for a multilayer perceptron (MLP) with a single hidden layer and soft plus activation function. Yet, obtaining closed-form bounds for neural networks with multiple hidden layers proved challenging. In their work, Weng et al. [1] introduced a novel robustness metric called Cross Lipschitz Extreme Value for network Robustness (CLEVER). This metric transformed the analysis of robustness into an estimation problem of local Lipschitz constants, providing a theoretical foundation. The CLEVER metric could be applied to any deep neural network classifier, leveraging extreme value theory to estimate the magnitude of the minimum perturbation required for generating effective adversarial samples. This approach offers a reliable assessment of robustness.

17.2.2 *Decision Boundary and Adversarial Samples*

Numerous studies have explored the vulnerability of trained neural networks to adversarial samples, with a focus on decision boundaries and their relationship to adversarial samples. Researchers have emphasized the significance of the decision boundary and its associated properties as crucial components of a classifier [3, 31]. In classification problems, the decision boundary of a deep neural network (DNN) reflects its ability to fit the data and generalize to unseen examples. Cortes and Vapnik [32] demonstrated that samples located near the decision boundary have a more substantial impact on the performance of the model compared to those located further away from it. This highlights the importance of understanding the behavior of the decision boundary in relation to the model's performance.

Moreover, the consistent success of powerful targeted adversarial attacks such as PGD [22] and C&W [26] has indicated that adversaries can generate adversarial perturbations to manipulate the classification output of a natural image. Tanay and Griffin [33] argued that adversarial samples occur when the classification boundary closely aligns with the sub-manifold of sampled data, although their analysis was confined to linear classifiers. Cao and Gong [9] discovered that adversarial samples tend to reside near the decision boundary and exploited this observation to develop defense mechanisms against adversarial attacks. He et al. [4] introduced a technique to examine properties of decision boundaries around an input instance, such as distances to the boundaries and neighboring classes. This approach aids in assessing the efficacy of adversarial attacks. Shamir et al. [34] explained the existence of adversarial samples based on the geometric structure of partitions defined by decision boundaries. Nevertheless, their work was confined to linear decision boundaries and did not consider the actual distance to the decision boundaries.

Jiang et al. [5] introduced an innovative approach to compute adversarial examples without relying on gradient information around the sample points. Their method generated improved samples that align closely with the decision boundaries of the model while requiring smaller perturbations. Karimi et al. [6] proposed a technique called Deep Decision boundary Instance Generation (DeepDIG), which leverages adversarial sample generation. They observed that the classification oscillates near the decision boundary and devised a novel metric to evaluate the robustness based on the oscillation between two classes. This metric provides insights into the model's susceptibility to adversarial attacks.

17.2.3 *Robustness Based on Adversarial Attack and Defense*

Assessing the robustness of a machine learning model often involves employing adversarial attacks, and the method of generating adversarial samples plays a crucial role in this evaluation. In the context of image classification models, these models are commonly evaluated based on the distortions measured in terms of p -norm in the pixel

feature space. Adversarial samples are typically created by constraining the l_0 , l_2 , and l_∞ norms of the perturbations, ensuring that they remain visually imperceptible to humans. This chapter primarily focuses on analyzing the model's robustness against l_2 and l_∞ attacks, instead of l_0 attacks, as explained in the subsequent discussion.

l_0 norm attacks aim to identify the minimum number of pixels that must be altered in order to deceive a classifier. Currently, there exist gradient-based white-box attacks [26, 35, 36], while black-box attacks predominantly employ local search or evolutionary algorithms [37–40]. However, l_0 norm attacks face limitations in terms of their applicability to different model types and dataset scales, resulting in limited research progress in this area.

Meanwhile, l_2 norm and l_∞ norm attacks have garnered significant research attention. l_∞ norm attacks focus on maximizing the absolute value of perturbations. A notable example is the Fast Gradient Sign Method (FGSM) proposed by Ian Goodfellow et al. [23]. Other works in this domain include Boosting Iterative Method (BIM) [41], Momentum Iterative Fast Gradient Sign Method (MI-FGSM) [41], Distributed Adversarial Attack (DAA) [42], and Transfer-Based Targeted Adversarial Attack [43].

On the other hand, l_2 norm attacks measure the Euclidean distance of perturbations [29]. There are methods that generate attacks using combinations of l_∞ , l_2 , and l_0 norms. For instance, the l_2 norm attack is often used alongside an l_∞ attack [22, 25, 26, 44].

17.3 Methodology

17.3.1 Preliminaries

Deep Neural Classifier. A deep neural network (DNN) is composed of input, hidden, and output layers. The strength of a deep classifier in accurately predicting results is typically not solely dictated by the model's structure, such as the number of hidden layers in the neural network. Given an input sample x , the hidden layer of the DNN produces the corresponding feature output, denoted as $h(x)$. The final classification label is determined by the output layer, represented as $f(x)$.

Consisting of the major component of a deep classifier except for the output layer, the embedding function $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$ maps the N -dimensions input x into the M -dimensional embedding space representation e . Generally, for a K-class $y \in \{1, \dots, K\}$ classification task $c(f(x)) \rightarrow y$, the classifier's output y can be obtained by the SoftMax value of its M -dimensional embedding result $f(x) : (e_1, \dots, e_M)$. Thus, we have the classification result as

$$y = \arg \max_k \left(\frac{\exp(c_k(f(x)))}{\sum_{k=1}^K \exp(c_k(f(x)))} \right). \quad (17.1)$$

Adversarial Robustness. When subjected to a well-crafted perturbation, an adversarial sample x^{adv} can effectively deceive the classifier, resulting in a wrong decision from $c(f(x)) \rightarrow y$ to $c(f(x^{adv})) \rightarrow y'$. In order to analyze the robustness of deep classifiers against adversarial attacks, we provide a general definition. Given a set of permissible perturbations $S \in \mathbb{R}^N$, a deep classifier is considered robust if it maintains the same decision outcome for the original input, denoted as

$$c(f(x + S)) \rightarrow y, \quad \text{while } c(f(x)) \rightarrow y. \quad (17.2)$$

Feature Vector. To represent the input x , we utilize its M-dimensional embedding $f(x) : (e_1, \dots, e_M)$ in the feature space. Let's assume that the classifier produces a decision result $y = k \in \mathbb{R}^K$, and we denote the feature vector of this sample as $f_{x,k}$. For the set of samples belonging to class k , denoted as N_k , we compute the center of this class as the mean vector of the embedded samples within the class. This center is represented as

$$f_{c_k} = \frac{1}{|N_k|} \sum_{x_i \in N_k} f_{x_i,k}. \quad (17.3)$$

Distance Metric. We use the Minkowski Distance [45] to measure the distance between a pair of samples represented as two M-dimensional vectors $f_{x_1} : (e_{11}, \dots, e_{1M})$ and $f_{x_2} : (e_{21}, \dots, e_{2M})$ in the feature space, written as

$$\text{dist}(f_{x_1}, f_{x_2}) = \left(\sum_a^M |e_{1a} - e_{2a}|^p \right)^{1/p}. \quad (17.4)$$

17.3.2 The Proposed Metrics for Robustness Evaluation

In our approach, we propose evaluating the robustness of the model by incorporating statistical features based on decision boundaries. This evaluation is conducted from two perspectives: feature subspace aggregation (FSA) within the same class and feature subspace distance (FSD) between different classes.

FSA focuses on the compactness of data within the same class. A higher degree of data aggregation indicates a smaller distance between samples within the same class in the feature space and the center of that class's features. Consequently, a model with greater data aggregation exhibits increased robustness.

$$FSA_k = 1 - \frac{\text{norm}(\sum_{j=1}^{n_k} \text{dist}(f_{x_j,k}, f_{c_k}))}{n_k} \quad (17.5)$$

$$FSA = \frac{\sum_{i=1}^K FSA_i}{K}, \quad (17.6)$$

where n_k represents the number of samples belonging to the k th class in the dataset. The function $norm(\cdot)$ indicates the process of standardization. The term $f_{x_j,k}$ corresponds to the feature vector of the sample x_j belonging to the k th class in the high-dimensional feature space. The notation f_{c_k} represents the center of the k th class. Here, K denotes the total number of classes. To calculate the overall FSA, we compute the average sum of all FSA_k values for each class within the set of K classes.

The feature subspace distance (FSD) characterizes the differentiation between different classes. A higher average distance between all classes suggests a greater robustness of the model.

$$FSD_{k,k+1} = dist(f_{c_k}, f_{c_{k+1}}) \quad (17.7)$$

$$FSD = \frac{\sum_{i=1}^{K-1} \sum_{j=i+1}^K FSD_{i,j}}{K(K-1)/2}, \quad (17.8)$$

where f_{c_k} and $f_{c_{k+1}}$ denote the center of the feature subspace of the k th and $k+1$ th data. We calculate the final FSD by averaging the sum of all $FSD_{i,j}$ between each class.

Our method combines the feature subspace aggregation (FSA) and feature subspace distance (FSD) into a unified metric. A smaller value of our method implies a reduced overlap between different feature subspaces, a larger distance between decision boundaries, and a higher level of robustness in the deep model.

$$ourmethod_{k,k+1} = FSA_k + FSA_{k+1} - FSD_{k,k+1} \quad (17.9)$$

$$ourmethod = \frac{\sum_{i=1}^{K-1} \sum_{j=i+1}^K ourmethod_{i,j}}{K(K-1)/2}, \quad (17.10)$$

where FSA_k and FSA_{k+1} represent the feature subspace aggregation of the k th and $k+1$ th data, and $FSD_{k,k+1}$ represents the feature subspace distance of the k th and $k+1$ th data. Finally, we calculate our method by averaging over all $ourmethod_{i,j}$ between classes.

Due to our choice of using the l_p -norm distance in the calculation of our method metric, the resulting metrics can take the form of l_p . Additionally, the time complexity of our method metrics is $O(n^2)$, where n represents the number of samples.

17.3.3 The Key Properties of Our Method

With our method's implementation algorithm above, we highlight our method's key properties are as follows:

Property 1: Smaller metric reflects higher robustness of the model. Referring to Eqs. 17.9 and 17.10, a smaller value of our method signifies a higher degree of data aggregation within each class. Simultaneously, it implies a greater distance between different classes, resulting in reduced overlap among them. Consequently, each sample tends to be situated further away from the decision boundary and other classes. This indicates that generating adversarial samples would require more significant perturbations.

Property 2: Our metric can be calculated only with natural samples, instead of generating adversarial samples. To determine the data distribution, we use M-dimensional embeddings of each input as their representation in the feature space. This approach ensures that the data distribution calculation is independent of any potential attacks.

Property 3: Our metric is model-agnostic, applicable to natural models, defensive models, and models with varying complexity. Based on Eq. 17.1, a deep classifier can be expressed as an embedding function followed by a SoftMax function. To calculate our method, we only require the embedding output and classification result of the model. These values are straightforward to obtain and do not depend on the specific model architecture, making our method model-agnostic.

17.4 Experiments

In this section, we evaluate the robustness evaluation metrics on various deep models, considering different settings and model capacities. The experiments are organized to address the following research questions:

Q1: How accurate can our method quantify the model’s robustness measured by attack success rate (ASR) when compared with baselines? In Sect. 17.4.2, we conduct experiments on white-box attacks (specifically, PGD) with varying attack parameters. We compare our method with baseline approaches to analyze the correlation and strength between ASR and our method. Additionally, we also perform experiments on black-box attacks (specifically, boundary attacks) to further evaluate the effectiveness of our method.

Q2: Can our method evaluate robust models of defended networks against adversarial samples? In Sect. 17.4.3, we evaluate the effectiveness of our method on robust models of defended networks against adversarial samples. For this purpose, we conduct experiments on both natural and robust models. We input original samples and adversarial samples to examine the performance of our method in each case.

17.4.1 Setup

Datasets and Models. We conduct experiments on MNIST, CIFAR-10, CIFAR-100 and Tiny-ImageNet. We evaluate the robustness metrics on the state-of-the-art neural

network models, including DenseNet [13], MobileNetV1 [14], MobileNetV2 [15], ResNet50 [16], ResNet101 [16], InceptionV1 [17], InceptionV2 [18], AlexNet [19], SqueezeNet [20], LeNet [21] (LeNet-5), simple fully connected network, and convolutional network. During the training process, we utilize the same training set for each dataset. Subsequently, we employ 10,000 test images to calculate the metrics for our method. To ensure a fair comparison, all models converge to natural models after training and achieve similar classification accuracy. The metrics for our method and the attack success rate are calculated under identical settings. The classification accuracies of the models are presented in the first column of Fig. 17.1.

Attack Methods. To evaluate the robust metrics, we consider both white-box and black-box attacks. For the white-box attack, we adopt the state-of-the-art first-order attack method known as sign-based Projected Gradient Descent (PGD). We evaluate both l_2 and l_∞ norm PGD attacks. In the l_∞ bounded PGD attack, we take gradient steps, while in the l_2 -bounded PGD attack, we use the gradient direction. For the MNIST dataset, we run 40 iterations with a step size of 0.01 and a perturbation size of $\epsilon = 0.3$. For CIFAR-10, CIFAR-100, and Tiny-ImageNet, the step size is chosen as 2, and the perturbation size is set to $\epsilon = 1.0$. For the black-box attack, we employ the boundary attack [46] on both MNIST and CIFAR-10 datasets to evaluate our method. We run 5,000 iterations on each dataset, with a sample size of 20 and a perturbation size of $\epsilon = 0.3$.

Defense Methods. As a defense method, we employed the min-max robust optimization-based adversarial training [22], which has been widely used to improve adversarial robustness in model compression tasks [47, 48]. For l_∞ adversarial training, we used an equal number of original samples and adversarial samples. The adversarial samples were generated under the same settings mentioned earlier.

Notations and Baseline Metrics. To evaluate the classification accuracy of the model and the attack success rate, we utilize ACC (classification accuracy) and ASR (attack success rate) metrics. Pearson correlation coefficient is employed to measure the strength of the correlation between these metrics. Additionally, as our baselines, we consider the ER (error rate) [25] and CLEVER (cross-Lipschitz extreme value for estimating robustness) scores [1] for evaluating the robust boundary of the model.

① Classification accuracy: $ACC = \frac{n_{true}}{N}$, where n_{true} is the number of original samples correctly classified by the targeted model and N denotes the total number of original images. The model with higher ACC is equipped with stronger classification capability.

② Attack success rate: $ASR = \frac{N_{adv}}{N}$, where N_{adv} denotes the number of samples misclassified by the targeted model after attacks. Higher ASR means that the model is more vulnerable to attacks.

③ Pearson correlation coefficient:

$$p = \frac{\sum_{i=1}^N (x - \bar{x})(y - \bar{y})}{[\sum_{i=1}^N (x - \bar{x})^2 \sum_{i=1}^N (y - \bar{y})^2]^{\frac{1}{2}}},$$

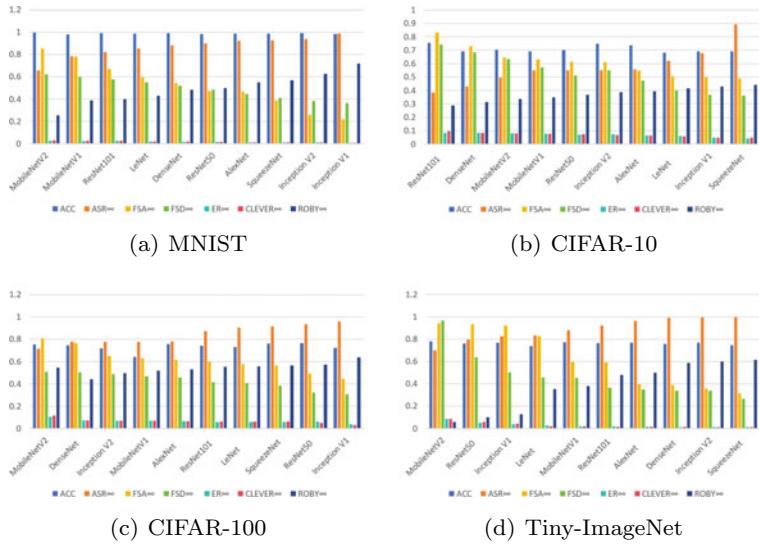


Fig. 17.1 Robustness evaluation based on ASR, ROBY, and baselines (l_∞ norm)

where x and y are the two variables whose correlations are measured. The variable \bar{x} represents the mean value of x , while \bar{y} represents the mean value of y . These values are used to quantify the similarity between vectors. A larger absolute value of p indicates a stronger correlation between the two vectors.

④ Empirical robustness (ER) [25]: The ER computes the robust perturbation size(σ, ϵ), which makes it satisfy as

$$\forall x_1, x_2 \in R, \|x_1 - x_2\|_p \leq \sigma \Rightarrow \|f(x_1) - f(x_2)\| \leq \epsilon, \quad (17.11)$$

where R is the input region and $f(\cdot)$ is the output function of the model.

⑤ CLEVER scores [1]: The CLEVER computes the lowest robust boundary L_{p,x_0}^j through sampling a set of points x^j in a ball $\mathbb{B}_p(x_0, R)$ around an input x_0 and takes the maximum value of $\|\Delta g(x_0)\|_p$, which is calculated as

$$L_{p,x_0}^j = \max_{x \in \mathbb{B}_p(x_0, R)} \|\Delta g(x_0)\|_p, \quad (17.12)$$

where $g(x_0) = f_c(x_0) = f_j(x_0)$ and c, j are two different classes.

In our experiment, both ER and CLEVER use 10,000 images from the test dataset for calculation. N_h and N_s of CLEVER are chosen as 500 and 1,000, respectively.

17.4.2 Benchmark Robustness Measured by Our Metric on Deep Models

In this subsection, we conduct experiments to answer **Q1**.

To ensure a comprehensive evaluation of our method's consistency with the Attack Success Rate (ASR), we calculate two variants of our method metrics as well as the ASR for the l_∞ and l_2 -norm PGD attacks on each natural model. For a fair comparison, we also compute the Error Rate (ER) (ER_∞, ER_2) and the CLEVER score ($CLEVER_\infty, CLEVER_2$). The l_∞ and l_2 forms of our method metrics are represented as $FSA_\infty, FSD_\infty, ourmethod_\infty$, and $FSA_2, FSD_2, ourmethod_2$, respectively. The l_∞ and l_2 norm ASRs of the PGD attack are denoted as ASR_∞ and ASR_2 , respectively.

17.4.2.1 Robustness Evaluation Compared with Baselines

To provide an overview of the robustness of models on each dataset, we rank the models in descending order based on their ASR. Once the robustness evaluation is established using ASR, we compare our method metrics, ER, and CLEVER scores with their respective ASR in the corresponding norm form. Specifically, we compare $ER_\infty, CLEVER_\infty, FSA_\infty, FSD_\infty, ourmethod_\infty$ with ASR_∞ , and $ER_2, CLEVER_2, FSA_2, FSD_2, ourmethod_2$ with ASR_2 . Detailed results can be found in Fig. 17.1.

As anticipated, our method metrics for each model align well with their corresponding ASR. Models that exhibit higher robustness against l_∞ attacks tend to have lower ASR_∞ values. Consequently, they tend to have larger FSA_∞ and FSD_∞ values, and smaller $ourmethod_\infty$ values. The same trend can be observed for l_2 attacks across all five datasets.

In terms of the baselines, the Error Rate (ER) occasionally displays inconsistencies with the ASR ranking when the ASR values are very close. For instance, on the MNIST dataset, the ASR_∞ of AlexNet is higher than that of SqueezeNet, but the ER_∞ of SqueezeNet is higher than that of AlexNet. This phenomenon indicates the potential inaccuracies in the evaluation of ER. Conversely, CLEVER scores, which are based on the maximum estimation theory, exhibit strong consistency with ASR.

17.4.2.2 Correlation Strength Analysis Between Metrics and ASR

The previous experiments demonstrate that the Error Rate (ER) is inferior to both our method and CLEVER in terms of consistency with ASR. To further compare the correlation between the CLEVER score and our method with ASR, we utilize the Pearson correlation coefficient to measure the strength of these correlations. The results are presented in Fig. 17.2.

The table reveals that both $CLEVER_\infty$ and $CLEVER_2$ exhibit negative correlations with ASR_∞ and ASR_2 across the five datasets. Notably, the absolute values of $ourmethod_\infty$ and $ourmethod_2$ are the highest among the five datasets, indicating

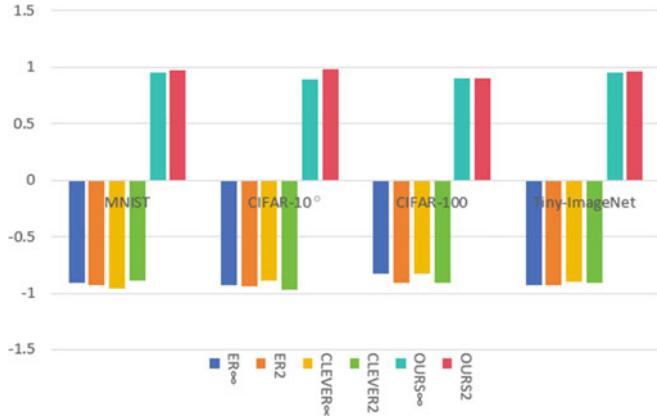


Fig. 17.2 Pearson correlation coefficient value of metrics (l_∞ norm, l_2 norm) and ASR (ASR_∞ , ASR_2)

Table 17.1 The relationship between ASR_∞ , ASR_2 and the our method metrics ($ourmethod_\infty$, $ourmethod_2$) under different perturbation sizes ϵ

Perturbation	Metrics	1	2	3	4	5	6	7	8
$ASR_\infty, 0.2$	OURS	0.25	0.39	0.40	0.43	0.48	0.50	0.55	0.57
	ASR	0.48	0.50	0.62	0.68	0.71	0.73	0.78	0.80
$ASR_\infty, 0.3$	OURS	0.25	0.39	0.40	0.43	0.48	0.50	0.55	0.57
	ASR	0.65	0.77	0.81	0.84	0.86	0.88	0.90	0.91
$ASR_\infty, 0.4$	OURS	0.25	0.39	0.40	0.43	0.48	0.50	0.55	0.57
	ASR	0.70	0.80	0.82	0.88	0.91	0.94	0.97	0.98
$ASR_2, 0.2$	OURS	0.29	0.41	0.45	0.52	0.56	0.59	0.61	0.62
	ASR	0.80	0.81	0.82	0.83	0.84	0.85	0.87	0.88
$ASR_2, 0.3$	OURS	0.29	0.41	0.45	0.52	0.56	0.59	0.61	0.62
	ASR	0.88	0.90	0.91	0.92	0.93	0.94	0.94	0.95
$ASR_2, 0.4$	OURS	0.29	0.41	0.45	0.52	0.56	0.59	0.61	0.62
	ASR	0.92	0.93	0.94	0.95	0.95	0.96	0.97	0.98

that our method achieves the highest Pearson correlation coefficient values. Consequently, our method outperforms the CLEVER score in reflecting the model's performance. The robustness of the models against l_∞ attacks, as well as the corresponding FSA_∞ and FSD_∞ values, displays positive correlations and negative correlations with $ourmethod_\infty$, respectively. Similarly, the robustness of the models against l_2 attacks, along with the associated FSA_2 and FSD_2 values, is positively correlated and negatively correlated with $ourmethod_2$, respectively. Thus, we recommend using $ourmethod_\infty$ and $ourmethod_2$ as better evaluation metrics for further experiments (Table 17.1).

17.4.2.3 Analysis of Robustness Measurement Against Black-box Attacks

The aforementioned experiments have demonstrated that our method can effectively serve as a substitute for the Attack Success Rate (ASR) in white-box attacks. In order to further analyze the relationship between the ASR of the boundary attack and our method, we aim to establish the consistency of our method in measuring the model's robustness. For this analysis, we specifically focus on the l_2 norm version of our method, as the boundary attack calculates the Euclidean distance predicted by the model.

Considering that the ASR of the black-box attack follows the same trend as that of the white-box attack, we observe that models with higher values for our method exhibit lower resistance to attacks. This finding leads us to conclude that our method serves as a robustness evaluation metric that is immune to attacks, and it aligns well with the ASR.

Here, we can answer **Q1**: our method demonstrates a strong alignment with ASR across various attack parameters, surpassing the performance of the baselines. The correlation between our method and ASR is notably stronger, indicating that our method can accurately quantify the robustness of the model without the need to generate adversarial examples.

17.4.3 Evaluation of Robustness on Defensive Models

Previous research [12] has demonstrated that adversarial training procedures based on l_∞ attacks can effectively increase the minimum l_2 distance between training and test samples and the decision boundary. This notion of transferability is similar to the remarkable effectiveness of our method, specifically *ourmethod*₂, in evaluating model robustness against l_∞ adversarial attacks. To address **Q2** and further analyze the relationship between our method and model robustness, we computed *ourmethod*₂ separately for both natural and robust models using both original and adversarial samples.

In the subsequent experiments, we will utilize *ourmethod*₂ as a representation of our method metrics, which will be referred to as our method. Additionally, we will use ASR_∞ to denote the attack success rate, which will be referred to as ASR. Tables 17.2 and 17.3 depict a comparison of our method metrics on both natural and robust defensive models.

When considering the general our method metric calculated with original samples, we observe that robust models consistently have smaller our method values compared to natural models across all five datasets. This finding confirms the effectiveness of our method in evaluating robustness, as l_∞ -norm PGD adversarial training proves to be a promising defense mechanism. Additionally, an intriguing result is observed when considering the our method metric calculated with adversarial samples. In this

Table 17.2 The comparison of our method metric between natural models and robust models on original samples

Model	$MNIST_N$	$MNIST_R$	$CIFAR10_N$	$CIFAR10_R$	$CIFAR100_N$	$CIFAR100_R$	$Tiny_N$	$Tiny_R$
MobileNetV2	0.29	0.28	0.34	0.35	0.42	0.41	0.07	0.06
MobileNetV1	0.41	0.37	0.38	0.34	0.52	0.46	0.38	0.37
ResNet101	0.45	0.44	0.22	0.21	0.56	0.48	0.49	0.36
LeNet	0.52	0.49	0.52	0.49	0.56	0.50	0.24	0.22
DenseNet	0.57	0.56	0.32	0.31	0.44	0.43	0.52	0.51
ResNet50	0.60	0.55	0.42	0.42	0.58	0.52	0.08	0.07
AlexNet	0.61	0.59	0.48	0.41	0.53	0.42	0.52	0.46
SqueezeNet	0.61	0.51	0.65	0.60	0.57	0.52	0.64	0.59
Inception V2	0.65	0.63	0.42	0.41	0.50	0.44	0.57	0.54
Inception V1	0.74	0.57	0.55	0.54	0.66	0.56	0.17	0.15

Table 17.3 The comparison of our method metric between natural model and robust model on adversarial samples

Model	$MNIST_N$	$MNIST_R$	$CIFAR10_N$	$CIFAR10_R$	$CIFAR100_N$	$CIFAR100_R$	$Tiny_N$	$Tiny_R$
MobileNetV2	0.29	0.28	0.34	0.35	0.42	0.41	0.07	0.06
MobileNetV1	0.41	0.37	0.38	0.34	0.52	0.46	0.38	0.37
ResNet101	0.45	0.44	0.22	0.21	0.56	0.48	0.49	0.36
LeNet	0.52	0.49	0.52	0.49	0.56	0.50	0.24	0.22
DenseNet	0.57	0.56	0.32	0.31	0.44	0.43	0.52	0.51
ResNet50	0.60	0.55	0.42	0.42	0.58	0.52	0.08	0.07
AlexNet	0.61	0.59	0.48	0.41	0.53	0.42	0.52	0.46
SqueezeNet	0.61	0.51	0.65	0.60	0.57	0.52	0.64	0.59
Inception V2	0.65	0.63	0.42	0.41	0.50	0.44	0.57	0.54
Inception V1	0.74	0.57	0.55	0.54	0.66	0.56	0.17	0.15

case, the robust model exhibits an even smaller our method value compared to the natural model, and experiences a more significant decline compared to the general our method.

The figure illustrates that both the natural and robust models have smaller our method values for both original and adversarial samples. Furthermore, it is worth noting that the natural and robust models generally have higher our method values for adversarial samples compared to original samples. There are several factors contributing to these observations. The robust model tends to have a more compact feature spatial distribution compared to the natural model for original samples, resulting in a larger distance between the two classes. This phenomenon effectively explains the smaller our method value of the robust models. As for adversarial samples, the feature distribution becomes concentrated near the decision boundary, making it more challenging for the models to distinguish between classes. The blurring of the feature space increases the likelihood of coincidental correct classifications, thus directly increasing the our method value. Models that have undergone adversarial

training can accurately classify adversarial samples due to clear decision boundaries. Consequently, the robust model exhibits a smaller our method value for adversarial samples compared to the natural model.

Here, we can answer **Q2**: our method can evaluate robust models of defended networks against adversarial samples. The our method value of the robust model is significantly lower than that of the natural model, regardless of whether the input sample is original or adversarial.

17.5 Conclusion

This chapter introduces our method, a novel and attack-independent measure of robustness that is based on the decision boundaries of models. Instead of relying on adversarial samples, our method utilizes inter-class and intra-class statistical features to depict the decision boundaries and evaluate the robustness of neural network classifiers. Through extensive experiments, we have demonstrated that our method aligns well with the attack-based robustness indicator ASR across various natural and defended networks. Moreover, our method is applicable to a wide range of state-of-the-art neural network classifiers and requires less computational resources compared to existing approaches for robustness evaluation.

References

1. Weng, T.W., Zhang, H., Chen, P.Y., Yi, J., Su, D., Gao, Y., Hsieh, C.J., Daniel, L.: Evaluating the robustness of neural networks: An extreme value theory approach. In: International Conference on Learning Representations (2018)
2. Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., Vechev, M.: Ai2: Safety and robustness certification of neural networks with abstract interpretation. In: 2018 IEEE symposium on security and privacy (SP). pp. 3–18. IEEE (2018)
3. Spangher, A., Ustun, B., Liu, Y.: Actionable recourse in linear classification. In: Proceedings of the 5th Workshop on Fairness, Accountability and Transparency in Machine Learning (2018)
4. He, W., Li, B., Song, D.: Decision boundary analysis of adversarial examples. In: International Conference on Learning Representations (2018)
5. Jiang, H., Song, Q., Le Kernec, J.: Searching the adversarial example in the decision boundary. In: 2020 International Conference on UK-China Emerging Technologies (UCET). pp. 1–4. IEEE (2020)
6. Karimi, H., Derr, T., Tang, J.: Characterizing the decision boundary of deep neural networks. arXiv preprint [arXiv:1912.11460](https://arxiv.org/abs/1912.11460) (2019)
7. Singla, S., Feizi, S.: Robustness certificates against adversarial examples for relu networks. arXiv preprint [arXiv:1902.01235](https://arxiv.org/abs/1902.01235) (2019)
8. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: Automated whitebox testing of deep learning systems. In: proceedings of the 26th Symposium on Operating Systems Principles. pp. 1–18 (2017)
9. Cao, X., Gong, N.Z.: Mitigating evasion attacks to deep neural networks via region-based classification. In: Proceedings of the 33rd Annual Computer Security Applications Conference. pp. 278–287 (2017)

10. Kim, B., Seo, J., Jeon, T.: Bridging adversarial robustness and gradient interpretability. arXiv preprint [arXiv:1903.11626](https://arxiv.org/abs/1903.11626) (2019)
11. Zha, Y., Ku, T., Li, Y., Zhang, P.: Deep position-sensitive tracking. IEEE Trans. Multimedia **22**(1), 96–107 (2019)
12. Mickisch, D., Assion, F., Greßner, F., Günther, W., Motta, M.: Understanding the decision boundary of deep neural networks: An empirical study. arXiv preprint [arXiv:2002.01810](https://arxiv.org/abs/2002.01810) (2020)
13. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4700–4708 (2017)
14. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint [arXiv:1704.04861](https://arxiv.org/abs/1704.04861) (2017)
15. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR [abs/1801.04381](https://arxiv.org/abs/1801.04381) (2018)
16. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
17. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
18. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning. pp. 448–456. PMLR (2015)
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Adv. Neural. Inf. Process. Syst. **25**, 1097–1105 (2012)
20. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and 0.5 mb model size. arXiv preprint [arXiv:1602.07360](https://arxiv.org/abs/1602.07360) (2016)
21. LeCun, Y., et al.: Lenet-5, convolutional neural networks. URL: <http://yannlecun.com/exdb/lenet> **20**(5), 14 (2015)
22. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net (2018)
23. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: ICML (2015)
24. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world. arXiv preprint [arXiv:1607.02533](https://arxiv.org/abs/1607.02533) (2016)
25. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 2574–2582 (2016)
26. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: 2017 ieee symposium on security and privacy (sp). pp. 39–57. IEEE (2017)
27. Wang, B., Gao, J., Qi, Y.: A theoretical framework for robustness of (deep) classifiers under adversarial noise. arXiv preprint [arXiv:1612.00334](https://arxiv.org/abs/1612.00334) (2016)
28. Gopinath, D., Katz, G., Păsăreanu, C.S., Barrett, C.: Deepsafe: A data-driven approach for assessing robustness of neural networks. In: International Symposium on Automated Technology for Verification and Analysis. pp. 3–19. Springer (2018)
29. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., Fergus, R.: Intriguing properties of neural networks. arXiv preprint [arXiv:1312.6199](https://arxiv.org/abs/1312.6199) (2013)
30. Hein, M., Andriushchenko, M.: Formal guarantees on the robustness of a classifier against adversarial manipulation. In: Advances in Neural Information Processing Systems. pp. 2266–2276 (2017)

31. Vemuri, V.R.: Artificial neural networks: theoretical concepts. IEEE Computer Society Press (1988)
32. Cortes, C., Vapnik, V.: Support-vector networks. *Machine learning* **20**(3), 273–297 (1995)
33. Tanay, T., Griffin, L.: A boundary tilting persepective on the phenomenon of adversarial examples. arXiv preprint [arXiv:1608.07690](https://arxiv.org/abs/1608.07690) (2016)
34. Shamir, A., Safran, I., Ronen, E., Dunkelman, O.: A simple explanation for the existence of adversarial examples with small hamming distance. arXiv preprint [arXiv:1901.10861](https://arxiv.org/abs/1901.10861) (2019)
35. Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: 2016 IEEE European symposium on security and privacy (EuroS&P). pp. 372–387. IEEE (2016)
36. Modas, A., Moosavi-Dezfooli, S.M., Frossard, P.: Sparsefool: a few pixels make a big difference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9087–9096 (2019)
37. Narodytska, N., Kasiviswanathan, S.P.: Simple black-box adversarial perturbations for deep networks. arXiv preprint [arXiv:1612.06299](https://arxiv.org/abs/1612.06299) (2016)
38. Schott, L., Rauber, J., Bethge, M., Brendel, W.: Towards the first adversarially robust neural network model on mnist. In: Seventh International Conference on Learning Representations (ICLR 2019). pp. 1–16 (2019)
39. Croce, F., Hein, M.: Sparse and imperceptible adversarial attacks. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 4724–4732 (2019)
40. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **23**(5), 828–841 (2019)
41. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 9185–9193 (2018)
42. Zheng, T., Chen, C., Ren, K.: Distributionally adversarial attack. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 2253–2260 (2019)
43. Inkawich, N., Wen, W., Li, H.H., Chen, Y.: Feature space perturbations yield more transferable adversarial examples. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7066–7074 (2019)
44. Chen, P.Y., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.J.: Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security. pp. 15–26 (2017)
45. Van de Geer, J.P.: Some aspects of Minkowski distance. Leiden University, Department of Data Theory (1995)
46. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. arXiv preprint [arXiv:1712.04248](https://arxiv.org/abs/1712.04248) (2017)
47. Ye, S., Xu, K., Liu, S., Cheng, H., Lambrechts, J.H., Zhang, H., Zhou, A., Ma, K., Wang, Y., Lin, X.: Adversarial robustness vs. model compression, or both? In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 111–120 (2019)
48. Gui, S., Wang, H.N., Yang, H., Yu, C., Wang, Z., Liu, J.: Model compression with adversarial robustness: A unified optimization framework. In: Advances in Neural Information Processing Systems. pp. 1285–1296 (2019)

Chapter 18

Certifiable Prioritization for Deep Neural Networks via Movement Cost in Feature Space



18.1 Introduction

There are several prioritization methods, mainly including four aspects, i.e., coverage-based [1–3], surprise-based [4–6], confidence-based [7–9], and mutation-based [10] methods. The first two methods prioritize test inputs based on DNNs' neuron coverage and surprise adequacy activation traces, respectively. Confidence-based methods identify bug-revealing inputs by measuring the classifier's output probabilities. Mutation-based methods design a series of mutation operations, and then analyze the mutated output probabilities based on supervised learning. While these methods have made significant advancements in identifying inputs that reveal bugs earlier, they still encounter certain challenges.

Firstly, one of the main drawbacks of existing methods is the lack of formal guarantees, which makes them vulnerable to malicious attacks, i.e., prioritizing bug-revealing inputs at the back when attacked. More specifically, when using neuron activation suppression as an optimization objective, an adversary has the ability to create imperceptible adversarial perturbations [11, 12]. These bug-revealing inputs with perturbations will be prioritized at the back owing to low neuron coverage or inadequate activation traces, i.e., coverage-based and surprise-based methods lose their effectiveness. Similarly, confidence-based methods cannot work when increasing the highest probability value [13, 14] with adding well-designed perturbations to inputs. Mutation-based methods include input mutation and model mutation, both of which are similar to data augmentation [15, 16] and network modification [17, 18], respectively. Once the mutation operations are leaked, the adversary can bypass these operations by crafting malicious bug-revealing inputs. Then, mutation-based methods will be invalid. Therefore, a formal robustness guarantee for certifiable prioritization is required.

Secondly, almost all methods either suffer from prioritization effectiveness or efficiency issues. For example, coverage-based methods have been demonstrated to be ineffective and time-costly [7]. Surprise-based methods improve test input

prioritization by utilizing more advanced metrics (e.g., surprise adequacy and activation frequency), but are computationally expensive due to more parameter tuning. Confidence-based methods apply output probabilities to perform fast and lightweight prioritization, and their effectiveness is better than the previous two. However, once adversarial [13] or poisoned [19] inputs are injected into the test dataset, their effectiveness drops largely. Mutation-based test input prioritization, PRIMA [10], is the state-of-the-art (SOTA) method for DNNs, outperforming the confidence-based methods by an average of 10%, but with a time cost increase of more than 100 times.

Thirdly, most methods suffer from the generalizability issue, including the generalizability of tasks, data forms, model structures, and application scenarios. For instance, confidence-based methods rely on DNNs' output probabilities, and thus may not be directly generalized to a regression task. Meanwhile, their prioritization effectiveness for sequential data form (e.g., text data [20]) has been demonstrated to drop by an average of 30% in the existing study [10]. Mutation-based methods could be generalized to various data and tasks, but specialized domain knowledge is required to design diverse data-specific (e.g., structured data [21] and graph data [22]) mutation strategies. Moreover, it is unclear whether their model mutation strategies can still perform well on other model structures, such as graph convolution network (GCN) [22]. Except for confidence-based methods, the other three types are all designed for white-box testing and require DNNs details.

Thus, our design goals are as follows: (1) we intend to take formal guarantee into account when designing a certifiable prioritization method; (2) we want the certifiability to serve prioritization effectiveness without degrading efficiency; (3) we plan to evaluate its generalizability.

One of the main contributions of DNNs is automatic feature extraction [23], which maps test inputs from the data space to the feature space. Based on the mapping ability, Zheng et al. [24] improved the robustness of DNNs by pushing the test input to the target position (i.e., class center) based on inverse perturbation. Further, we find that the inverse perturbation measures the movement cost of test inputs in feature space. However, the inverse perturbation [24] is obtained through iterative training, which is still empirical. To satisfy the certifiability requirement, we further derive a formal guarantee of the inverse perturbation with the Lipschitz continuity assumption [25].

According to the utility analysis and certifiability consideration, we design a certifiable prioritization technique which reduces the problem of measuring misbehavior probability to the problem of measuring the movement difficulty in feature space, i.e., the movement cost of the test inputs being close to or far from the class centers. Then, we compute the certifiable inverse perturbation based on the generalized extreme value theory (GEVT) [26]. Based on the formal robustness guarantee, our method is valid for identifying malicious bug-revealing inputs, as well as clean bug-revealing inputs, without degrading efficiency.

The main contributions are as follows.

- Through inverse perturbation analysis and measurement, we first implement a formal robustness guarantee for the movement cost, which provides a new perspective for measuring DNNs' misbehavior probability.
- Based on the formal guarantee, we propose an effective and efficient technique that leverages certifiability to facilitate prioritization effectiveness.
- To evaluate our method's generalizability, we conduct extensive experiments on various tasks, data, models, and scenarios, the results of which show the superiority of our method compared with previous works.

18.2 Related Works

To solve the labeling-cost problem in DNN testing, several works on test input prioritization are proposed [4–10, 10, 27, 28].

From the perspective of statistical analysis, there are coverage-based [1–3, 29], surprise-based [4–6, 27], and confidence-based methods [7–9]. Feng et al. [7] comprehensively analyzed coverage-based methods and concluded that their effectiveness and efficiency are unsatisfactory for prioritization tasks. To improve effectiveness, Byun et al. [5] prioritized test inputs based on surprise adequacy metrics. Zhang et al. [4] observed the activation pattern of neurons, and produced prioritization results according to the activation patterns between the training set and test inputs. Ma et al. [27] considered the interaction between test inputs and model uncertainty, and determined bug-revealing inputs with higher uncertainty. These surprise-based methods improve performance, but their prioritization results are related to the training set quality. Furthermore, Zhang et al. [9] prioritized test inputs based on noise sensitivity analysis, independent of the training set. Shen et al. [8] proposed MCP, in which clusters test inputs into the boundary areas and specify the priority to select them evenly. Feng et al. [7] proposed DeepGini, which prioritizes test input by measuring set impurity. These confidence-based methods are effective and efficient, but only for classification tasks.

Drawing lessons from the mutation view in software engineering [30, 31], Wang et al. [10] proposed PRIMA, which gives priority to test inputs that generate different predictions through diversity mutations (i.e., input level and model level). PRIMA demonstrates SOTA performance, but cannot be applied to black-box scenarios.

18.3 Background

In this section, we first introduce the basic knowledge of DNNs, and then give the definitions of inverse perturbation.

18.3.1 Deep Neural Networks

DNN consists of several layers, each of which contains a large number of neurons[23]. Generally, the basic tasks of DNNs include classification and regression. The classification and regression models are as follows.

The classification model predicts which class the test input belongs to. Suppose we have a K -class classifier $f^C : \mathbb{R}^d \rightarrow \mathbb{R}^K$. Given a test input $x_0 \in \mathbb{R}^d$, the classifier will output a vector of K values normalized by SoftMax function [32], e.g., $\{f_i^C(x_0) | 1 \leq i \leq K\}$, each of which represents the probability that x_0 belongs to the i -th class, where $\{d, K\} \in \mathbb{Z}^+$ and $K \geq 2$. $c(x_0) = \arg \max_{1 \leq i \leq K} f_i^C(x_0)$ represents the predicted label of x_0 and $f_i^C(x_0) \in (0, 1)$.

The regression model describes a mapping between the test input and the output. Suppose we have a regression model $f^R : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$. Given a test input $x_0 \in \mathbb{R}^{d_1}$, the regression model will output a vector with d_2 elements activated by linear or ReLU functions [33], e.g., $\{f_i^R(x_0) | 1 \leq i \leq d_2\}$, each of which represents a fit to the ground truth, where $\{d_1, d_2\} \in \mathbb{Z}^+$. $r(x_0) = f^R(x_0)$ represents the fitted prediction output of x_0 and $f_i^R(x_0) \in [\min_r, \max_r]$.

18.3.2 Definitions of Inverse Perturbation

We give definitions of inversely perturbed test input, minimum inverse perturbation, and lower bound.

Definition 18.1 (*Inversely perturbed test input*) Given $x_0 \in \mathbb{R}^d$, we say x_0^* is an inversely perturbed test input of x_0 with inverse perturbation μ and l_p -norm Δ_p if $x_0^* = x_0 + \mu$ is moved to the target position and $\Delta_p = \|\mu\|_p$. An inversely perturbed test input for a classifier is $x_0^* \in \mathbb{R}^d$ that moves towards the class center of $c(x_0)$, where the class center is defined as $f_{center}^C(x_0) = \min \{f_c^C(x_0) \times [1 + \log(1 + f_c^C(x_0))], 1\}$. For a regression model, $x_0^* \in \mathbb{R}^{d_1}$ moves from $r(x_0)$ to regression center:

$$f_{i,+}^R(x_0) = \text{clip}_{\min_r}^{\max_r} (f_i^R(x_0) + |f_i^R(x_0)| \times \log [1 + \tanh(f_i^R(x_0))]). \quad (18.1)$$

Definition 18.2 (*Minimum inverse perturbation* Δ_p^{\min} and its lower bound γ_L) Given a test input x_0 , the minimum l_p inverse perturbation of x_0 , denoted as Δ_p^{\min} , is defined as the smallest Δ_p over all inversely perturbed test inputs of x_0 . Suppose Δ_p^{\min} is the minimum inverse perturbation of x_0 . A lower bound of Δ_p^{\min} , denoted by γ_L where $\gamma_L \leq \Delta_p^{\min}$, is defined such that any inversely perturbed test inputs of x_0 with $\|\mu\|_p \leq \gamma_L$ will never reach the target position.

The lower bound of inverse perturbation measures the minimum movement cost for a test input to reach the target position (i.e., class center or regression center). γ_L guarantees that the inversely perturbed test input will never move to the target position for inverse perturbation with $\|\mu\|_p \leq \gamma_L$, certifying the movement cost of the test input.

18.4 Methodology

In this section, we present a technical description of our method. First, we discuss the prioritization feasibility based on a movement cost view. Then, we introduce our method, which provides a formal robustness guarantee based on the Lipschitz continuity assumption [25] and estimates the movement cost based on GEVT [34]. Finally, we prioritize inputs via movement costs.

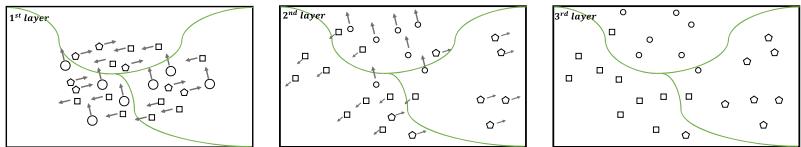
18.4.1 A Movement View in Feature Space

A well-trained DNN implements feature extraction through multiple hidden layers, each of which filters redundant features and amplifies key features during forward propagation [35]. If we regard the feature mapping of hidden layers as data movement in feature space, almost all test inputs are pushed towards the target position in forward propagation, while bug-revealing inputs fail to reach the target position at the end of forward propagation.

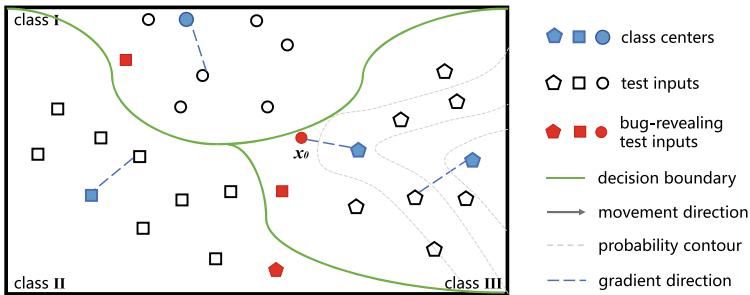
First, we investigate the movement process of test inputs in forward propagation. Taking a classifier based on a FCN with three hidden layers on MNIST [36] dataset as an example, the t-SNE [37]-based distribution of test inputs in feature space is visualized in Fig. 18.1. During the forward propagation as shown in Fig. 18.1a, we observe that most test inputs' directional approach the correct class, which realizes the right prediction. However, several test inputs move without direction leading to the DNN's misbehavior (i.e., misclassification), which are the bug-revealing inputs that need to be identified. It is intuitively based on the feature purity theory of test inputs [7]. For a bug-revealing test input, it not only contains features of multiple classes, but the highest feature purity is close to the second highest one, and even the feature purity of each class is close. Thus, the movement of this test input in the forward propagation is directionless.

Then, we further analyze the movement cost of correctly and incorrectly predicted test inputs based on the inverse perturbation [24]. As shown in Fig. 18.1b, the ground truth of x_0 is **II** but is misclassified as **III**. As x_0 proceeds along the gradient direction, its class center of **III** can be reached with low movement cost. However, correctly predicted test inputs require high movement cost to reach their class centers. It is consistent with the interpretation based on feature purity [7]. The incorrectly predicted test inputs improve the feature purity of the corresponding class after adding inverse perturbation, which turns random movement into directional movement in forward propagation. Therefore, only a low movement cost is required to reach the class center. The correctly predicted test inputs contain high feature purity, which keeps them stable in feature space. Thus, further movement requires a high cost. Note that the class center is given by Definition 18.1, which differs for each test input [24].

Based on the above analysis, we reduce the problem of measuring misbehavior probability of prioritization to the problem of measuring the movement cost in feature space. Thus, we prioritize test inputs by comparing their lower bounds of minimum



(a) The movement process of test inputs in forward propagation, where almost all test inputs move towards the corresponding class centers.



(b) The movement cost of test inputs in back propagation, where x_0 is a misclassified test input, “probability contour” represents that test inputs on the same contour line have the same probability of belonging to the class, “gradient direction” is obtained by back propagation [38].

Fig. 18.1 An example of t-SNE [37]-based distribution visualization of test inputs in feature space. The classifier is an FCN with three hidden layers, the test inputs are handwritten digits of “0”(I), “1”(II), and “2”(III) from MNIST dataset [36]

movement cost to the target position. The test input never reaches the target position when the movement cost is less than the lower bound γ_L . However, γ_L is not easy to find. Below we show how to derive a formal inverse perturbation guarantee of a test input with the Lipschitz continuity assumption [25].

18.4.2 Formal Guarantees for Movement Cost

We first give the lemma about the Lipschitz continuity. According to the lemma, we then provide a formal guarantee for the lower bound of the inverse perturbation. Specifically, our analysis obtains a lower bound of l_p -norm minimum inverse perturbation $\gamma_L = \min \frac{f_{center}^C(x_0) - f_c^C(x_0)}{L_q^c}$ for a classifier and $\gamma_L = \min \frac{\sum_i |f_{i,+}^R(x_0) - f_i^R(x_0)|}{d_2 \times L_q^r}$ for a regression model.

Lemma 18.1 (*Norms and corresponding Lipschitz constants [25]*). *Let $D \subset \mathbb{R}^d$ be a convex bounded closed set and let $h(x) : S \rightarrow \mathbb{R}$ be a continuously differentiable function on an open set containing D . For a Lipschitz function $h(x)$ with Lipschitz constant L_q , the inequality $|h(a) - h(b)| \leq L_q \|a - b\|_p$ holds for any $\{a, b\} \in D$,*

where $L_q = \max\{||\nabla_x h(x)||_q : x \in D\}$, $\nabla_x h(x) = (\frac{\partial h}{\partial x_1}, \dots, \frac{\partial h}{\partial x_d})$ is the gradient of $h(x)$, $\frac{1}{p} + \frac{1}{q} = 1$ and $1 \leq \{p, q\} \leq \infty$.

Theorem 18.1 (Formal guarantee on lower bound γ_L of inverse perturbation for classification model) Let $x_0 \in \mathbb{R}^d$ and $f^C : \mathbb{R}^d \rightarrow \mathbb{R}^K$ be a K -class classifier with continuously differentiable components. For all $\mu \in \mathbb{R}^d$ with $||\mu||_p \leq \min \frac{f_{center}^C(x_0) - f_c^C(x_0)}{L_q^c}$, $f_c^C(x_0 + \mu) = f_{center}^C(x_0)$ holds with $\frac{1}{p} + \frac{1}{q} = 1$, $1 \leq \{p, q\} \leq \infty$ and L_q^c is the Lipschitz constant for the function $f_{center}^C(x) - f_c^C(x)$ in l_p -norm. In other words, $\gamma_L = \min \frac{f_{center}^C(x_0) - f_c^C(x_0)}{L_q^c}$ is a lower bound of minimum inverse perturbation for moving to the class center.

Proof According to Lemma 1, the assumption that the function $h(x) := f_{center}^C(x) - f_c^C(x)$ is Lipschitz continuous with Lipschitz constant L_q^c gives

$$|h(a) - h(b)| \leq L_q^c ||a - b||_p. \quad (18.2)$$

Let $a = x_0 + \mu$ and $b = x_0$, we get

$$|h(x_0 + \mu) - h(x_0)| \leq L_q^c ||\mu||_p, \quad (18.3)$$

which can be rearranged by removing the absolute symbol:

$$\begin{aligned} -L_q^c ||\mu||_p &\leq h(x_0 + \mu) - h(x_0) \leq L_q^c ||\mu||_p, \\ \Rightarrow h(x_0) - L_q^c ||\mu||_p &\leq h(x_0 + \mu) \leq h(x_0) + L_q^c ||\mu||_p. \end{aligned} \quad (18.4)$$

When $h(x_0 + \mu) = 0$, the inversely perturbed test input is moved to the class center. As represented by Eq. (18.4), $h(x_0) - L_q^c ||\mu||_p$ is the lower bound of $h(x_0 + \mu)$. If $h(x_0) - L_q^c ||\mu||_p \geq 0$ for sufficiently small inverse perturbation $||\mu||_p$, the inversely perturbed input cannot reach the class center, i.e.,

$$\begin{aligned} h(x_0) - L_q^c ||\mu||_p &\geq 0 \Rightarrow ||\mu||_p \leq \frac{h(x_0)}{L_q^c} \\ \Rightarrow ||\mu||_p &\leq \frac{f_{center}^C(x_0) - f_c^C(x_0)}{L_q^c}. \end{aligned} \quad (18.5)$$

To realize $f_c^C(x_0 + \mu) = f_{center}^C(x_0)$ we take the minimum of the bound on $||\mu||_p$, i.e., the test input will never move to the class center when $||\mu||_p \leq \min \frac{f_{center}^C(x_0) - f_c^C(x_0)}{L_q^c}$.

Theorem 18.2 (Formal guarantee on lower bound γ_L of inverse perturbation for regression model) Let $x_0 \in \mathbb{R}^{d_1}$ and $f^R : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$ be a regression model with continuously differentiable components. For all $\mu \in \mathbb{R}^{d_1}$ with $||\mu||_p \leq \min \frac{\sum_i |f_{i,+}^R(x_0) - f_i^R(x_0)|}{d_2 \times L_q^r}$, $\frac{1}{d_2} \sum |r(x_0 + \mu) - r(x_0)| \leq \delta$ holds with $\frac{1}{p} + \frac{1}{q} = 1$, $1 \leq \{p, q\} \leq \infty$ and L_q^r is the Lipschitz constant for the function $\frac{\sum_i |f_{i,+}^R(x) - f_i^R(x)|}{d_2}$ in l_p -norm.

Algorithm 18.1: Our method

Input: Test inputs $\{x_i\}$, $i \in \{0, 1, \dots, N-1\}$, a classification model f^C , the sampling radius \mathcal{R} , batch size N_b , number of random samples per batch N_{rsb} , norm value p .
Output: The prioritization index Ω .

```

1  $h(x) := f_{center}^C(x) - f_c^C(x)$ ,  $q = \frac{p}{p-1}$ ,  $\Gamma = \{\emptyset\}$ .
2 For  $i = 0 : N-1$  do
3    $S_\nabla = \{\emptyset\}$ .
4   For  $j = 0 : N_b-1$  do
5     For  $k = 0 : N_{rsb}-1$  do
6        $g_{i,j,k} = \|\nabla_x h(x_i^{(j,k)})\|_q$ .
7     End For
8      $g_{i,j}^{\max} = \max\{g_{i,j,k}\}$  and  $S_\nabla = S_\nabla \cup \{g_{i,j}^{\max}\}$ .
9   End For
10  Estimate  $\hat{\xi}$  of Weibull distribution on  $S_\nabla$ .
11   $\gamma_L = \frac{h(x_i)}{-1/\hat{\xi}}$  and  $\Gamma = \Gamma \cup \gamma_L$ .
12 End For
13  $\Omega \leftarrow$  return the index of  $\Gamma$  in ascending order.
```

In other words, $\gamma_L = \min \frac{\sum_i |f_{i,+}^R(x_0) - f_i^R(x_0)|}{d_2 \times L_q^r}$ is a lower bound of minimum inverse perturbation.

18.4.3 Movement Cost Estimation via GEVT

The formal guarantees give that γ_L is related to $h(x_0)$ and its Lipschitz constant L_q , where $L_q = \max\{\|\nabla_x h(x)\|_q : x \in B_p(x_0, \mathcal{R})\}$, $B_p(x_0, \mathcal{R}) := \{x | \|x - x_0\|_p \leq \mathcal{R}\}$ is a hyper-ball with center x_0 and radius \mathcal{R} . The value of $h(x_0)$ is easily accessible at the model output. Thus, we show how to get $\max\|\nabla_x h(x)\|_q$.

For a random variable sequence $\{x_0^{(j)}\}$ sampled from $B_p(x_0, \mathcal{R})$, its corresponding gradient norm can be regarded as a new random variable sequence $\{\|\nabla_x h(x_0^{(j)})\|_q\}$ characterized by a cumulative distribution function (CDF) [34]. Therefore, we estimate $\max\|\nabla_x h(x_0^{(j)})\|_q$ with a small number of samples based on GEVT, which ensures that the maximum value of a random variable sequence can only follow one of the three extreme value distributions [26]. The CDF of $\|\nabla_x h(x_0^{(j)})\|_q$ is as follows:

$$G_\xi(z) = \exp\left(-(1 + \xi z)^{-\frac{1}{\xi}}\right), \quad (18.6)$$

where $1 + \xi z > 0$, $z = \frac{\|\nabla_x h(x_0^{(j)})\|_q - u}{\sigma}$, $\xi \in \mathbb{R}$ is an extreme value index, $u \in \mathbb{R}$ and $\sigma \in \mathbb{R}^+$ are the expectation and variance. The parameters u , σ , and ξ determine the location, scale, and shape of $G_\xi(z)$, respectively. $G_\xi(z)$ belongs to Weibull ($\xi < 0$) distributions, where the right end-point is the estimation of $\max\|\nabla_x h(x)\|_q$.

18.4.4 Prioritization Through Movement Cost

Based on the movement cost perspective in Sect. 18.4.1, we can conclude that a test input with a small value of γ_L should be prioritized at the front. Therefore, we compute the γ_L value of each test input and then prioritize these inputs according to their γ_L values from small to large. Algorithm 18.1 shows the details of our method by taking the classification model as an example. We first establish the gradient norm distribution of a test input by randomly sampling in the hyper-ball (the loop from lines 4 to 9). Then, we estimate the location of the maximum gradient norm based on the Weibull distribution and compute the lower bound of the movement cost based on Theorem 18.1 (lines 10 and 11). Finally, we repeat the above operations for each test input (the loop from lines 2 to 12) and prioritize them according to the ascending result of their γ_L values (line 13). Note that for algorithmic illustration, we only compute one $g_{i,j}^{\max}$ (line 8) for each iteration. To implement the best efficiency of GPU, we usually evaluate these values in batches, and thus a batch of $g_{i,j}^{\max}$ can be returned.

Furthermore, we can easily generalize our method to a regression model through replacing the $h(x)$ function at line 1 with $h(x) := \frac{\sum_i |f_{i,+}^R(x) - f_i^R(x)|}{d_2}$ based on Theorem 18.2. Besides, we can extend our method to a black-box scenario based on the gradient estimation [39].

18.5 Experimental Setting

In this section, we introduce the experimental setup, including the subjects we considered, the baselines we compared with, the measurements we used, and the implementation details we conducted.

18.5.1 Subjects

To sufficiently evaluate our method's effectiveness, efficiency, robustness, and generalizability, we carefully consider the diversity of subjects from six dimensions. To our best knowledge, this is the most large-scale and diverse study in the field.

- (1) **Various tasks of deep models.** We employ both classification models (ID: 1–32, 37–45, 47–49) and regression models (ID: 33–36, 46, 50). The number of classes ranges from 2 to 1,000 across all the classification tasks.
- (2) **Various data forms of test inputs.** We consider six data forms of test inputs from 14 datasets, including image, text, speech, signal, graph, and structured data. Specifically, we collect 6 *image datasets*, i.e., CIFAR10 [40], ImageNet [41], DrivingSA [42], Fashion-MNIST (FMNIST) [43], Ants_Bees (a binary insect dataset containing ants and bees for transfer learning), and Cats_Dogs (a binary pet dataset containing cats and dogs for transfer learning), 2 *text datasets*, i.e., IMDB [44] and Reuters (a 46-class newswire dataset), one *speech dataset*, i.e.,

VCTK10 (a 10-class dataset of ten English speakers with various accents selected from VCTK Corpus), *one signal dataset*, i.e., RML8PSK [45], *one graph dataset*, i.e., Cora [46], and *3 structured datasets*, i.e., Adult [47], COMPAS (a binary crime prediction dataset), and Boston (a housing price regression dataset).

- (3) **Various data types of test inputs.** We mainly consider three data types, including original test inputs, adversarial test inputs, and adaptive attacked test inputs. For *adversarial test inputs*, we perform three widely used adversarial attacks (i.e., basic iterative method (BIM) [48], Carlini and Wagner (C&W) [49], and FineFool [50]) to generate the same number of adversarial examples as the corresponding original test inputs for CIFAR10 and ImageNet, respectively. Then, following previous works [7, 10], we construct an adversarial test input set for each of the two datasets under each adversarial attack by randomly selecting half of the original test inputs and half of the adversarial examples, represented as “+BIM”, “+C&W”, and “+FineFool”. Regarding the surprise-based method [6], we consider two objectives, flipping the labels of test inputs, and reducing their surprise (i.e., the activation difference between test inputs and the training inputs), which can be conducted by MAG-GAN [13]. Regarding confidence-based methods [7, 8], we first flip the labels of test inputs based on BIM [48], and then increase their highest probability value (up to 0.99 for CIFAR10 and 0.90 for ImageNet) by continuously adding perturbations. Regarding the mutation-based method [10], we first break its ranking model by adding noise to the extracted features, and then add perturbations to the test inputs based on back propagation [38] to generate noisy features. Finally, we construct an adaptive attacked test input set for each of the two datasets under each adaptive attack by randomly selecting half of the original test inputs and half of the adaptive attacked examples, represented as “+AdapS”, “+AdapC”, and “+AdapM”. When crafting adversarial or adaptive attacked test inputs, we assume that the adversary knows the prioritization details.
- (4) **Various structures of deep models.** We employ CNN (ID: 1–41, 43), LSTM (ID: 42, 44–46), GCN (ID: 47), and FCN (ID: 48–50). The number of layers ranges from 3 to 101 across all models.
- (5) **Various training scenarios.** We set up three training scenarios, including normal training, poisoning, and transfer learning. For the *poisoning scenario*, we first craft a poisoned training set on FMNIST dataset, denoted as “FMNIST_P”, where the poisoning method is DeepPoison [52], the source label and poisoned label are “Pullover” and “Coat”, respectively. For the *transfer learning scenario*, we deploy the source domain as ImageNet with 1,000-class and the target domain as insect images or pet images with 2-class.
- (6) **Various prioritization scenarios.** We set up two prioritization scenarios, including white box and black box. All details of the model and test inputs are available and used for prioritization in the *white-box scenario* (ID: 1, 3–9, 11–17, 19–25,

27–33, 35–50), while only model outputs and test inputs are available in the *black-box scenario* (ID: 2, 10, 18, 26, 34).

18.5.2 Baselines

We consider five baselines, i.e., likelihood-based surprise adequacy (LSA) [6], distance-based surprise adequacy (DSA) [6], multiple-boundary clustering and prioritization (MCP) [8], DeepGini [7], and PRIMA [10]. LSA and DSA are surprise-based methods. MCP and DeepGini are efficient confidence-based methods for lightweight prioritization. PRIMA is an effective mutation-based method with the SOTA performance. Note that coverage-based methods have been shown to be significantly less effective [7] and thus are omitted. All baselines are configured according to the best performance setting reported in the respective papers.

18.5.3 Measurements

We investigate our method’s prioritization performance from five aspects, including prioritization *effectiveness*, *efficiency*, *robustness*, and *generalizability* (Fig. 18.2).

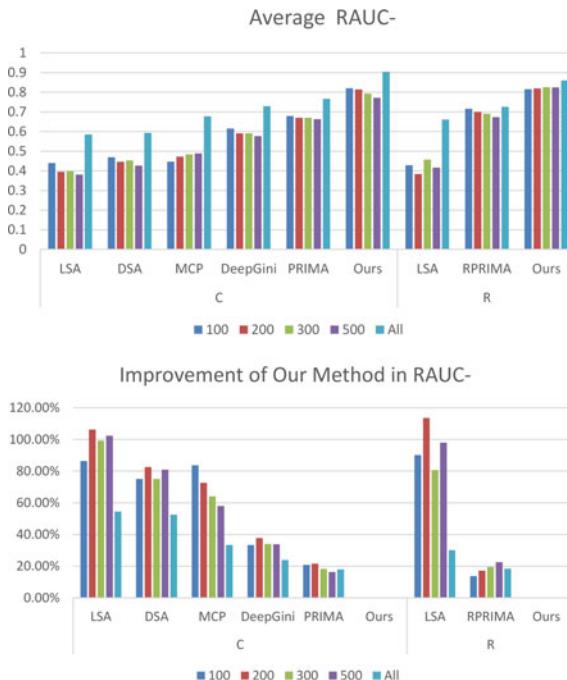


Fig. 18.2 Overall comparison results across all subjects, measured based on average RAUC and its improvement, where “C” and “R” represent classification and regression, “0” means not applicable in theory

- (1) We evaluate the effectiveness of our method through the ratio of the area under curve (RAUC), which transforms the prioritization result to a curve [10], defined as follows for classification tasks:

$$\text{RAUC} = \frac{\sum_{i=1}^N n_i}{N \times N' + \frac{N-N'^2}{2}}, \text{ where } n_i = \begin{cases} n_{i-1}+1, & c(x_i) \text{ is correct} \\ n_{i-1}, & \text{otherwise} \end{cases}, \quad (18.7)$$

where N and N' are the number of prioritized test inputs and bug-revealing inputs, $i \geq 1$ and $n_0=0$. In other words, the numerator and denominator represent the area under the curve of the prioritization method and the ideal prioritization, respectively.

For regression tasks, RAUC is calculated based on the mean square error (MSE) of prediction, as follows:

$$\text{RAUC} = \frac{\sum_{i=1}^N m_i}{\sum_{j=1}^N M_j}, \text{ where } \begin{cases} m_i = m_{i-1} + \text{MSE}(f^R(x_i)) \\ M_j = M_{j-1} + \text{MSE}(f^R(x_j)) \end{cases}, \quad (18.8)$$

where m_i and M_j represent the accumulated MSE between the predicted results and the ground truth of the prioritization method and the ideal prioritization, respectively. $i, j \geq 1$, $m_0=M_0=0$. Moreover, we follow the setup of Wang et al. [10], using RAUC-100, RAUC-200, RAUC-300, RAUC-500, and RAUC-all as fine-grained metrics, i.e., $N=100, 200, 300, 500$, and all. Larger RAUC is better.

- (2) We evaluate the prioritization efficiency of our method through prioritization speed, i.e., the time cost of prioritize 1,000 test inputs (#seconds/1,000 Inputs). Smaller is better.
- (3) We evaluate the prioritization robustness of our method through RAUC stability, denoted as RobR. Larger RobR is better, as follows:

$$\text{RobR} = \frac{\text{RAUC-all of adaptive attacked test inputs}}{\text{RAUC-all of original test inputs}} \times 100\%. \quad (18.9)$$

- (4) We evaluate the prioritization generalizability of our method based on reward value, denoted as GenRew, as follows:

$$\text{GenRew} = \frac{1}{N_{rep} \times N_{sub}} \sum_{i=1}^{N_{rep}} \sum_{j=1}^{N_{sub}} \frac{n_{pm} - k_{i,j} + 1}{n_{pm}}, \quad (18.10)$$

where N_{sub} represents the number of selected subjects, $N_{rep}=5$ and $n_{pm}=6$ are the number of repetitions and prioritization methods in our experiments, respectively. $k_{i,j} \in \{1, 2, \dots, n_{pm}\}$ represents that the method ranks in the k -th position in descending order of RAUC-all for the j -th subject at the i -th experimental repetition. $\text{GenRew} \in [\frac{1}{n_{pm}}, 1]$. Larger GenRew is better.

18.5.4 Implementation Details

To fairly study the performance of the baselines and our method, our experiments have the following settings. **(1) Hyperparameter settings** based on the double-minimum strategy: we conduct a preliminary study based on a small dataset, and find that $N_b > 3$, $N_{rsb} > 5$, and $0.02 \max(x) < \mathcal{R} < 0.05 \max(x)$ for Algorithm 18.1 are effective in general. To guarantee our method’s effectiveness, we follow the double-minimum strategy, i.e., $N_b=6$, $N_{rsb}=10$, $\mathcal{R}=0.04\max(x)$, $p=2$ for Algorithm 18.1. **(2) Model training:** we download and use pre-trained models on ImageNet. For other datasets, we train appropriate models as follows. The learning rate ranges from 1E-04 to 1E-02, the optimizer is Adam, training:validation:test = 7:1:2, and an early stop strategy is used to avoid overfitting. **(3) Preprocessing:** we fill in missing data points based on the mean value. There is no specific mutation strategy provided in PRIMA [10] for speech, signal, graph data forms, and GCN model, thus we derive it from the existing mutation operations. **(4) Data recording:** we repeat the experiment 5 times and record about 9,000 raw data.

18.6 Experimental Results and Analysis

We evaluate our method through answering the following research questions (RQ):

RQ1. Effectiveness: How *effective* is our method?

RQ2. Efficiency: How *efficient* is our method?

RQ3. Robustness: How *robust* is our method?

RQ4. Generalizability: How *generic* is our method?

18.6.1 Effectiveness (RQ1)

How *effective* is our method in prioritizing test inputs?

When reporting the results, we focus on the effectiveness of the following aspects: overall, data forms, data types, training scenarios, and prioritization scenarios. The evaluation results are shown in Tables 18.3, 18.4, Figs. 18.2, 18.5, 18.6, and 18.7.

Implementation details for effectiveness evaluation. (1) We present the overall comparison results in Fig. 18.2. Since DSA, MCP, and DeepGini cannot directly apply to regression tasks, we separately present results on both tasks. (2) We conduct a preliminary T-test about RAUC across all subjects, as shown in Fig. 18.3. (3) Since DeepGini and PRIMA show significantly better performance than other baselines, we mainly compare the results between our method and these two in Table 18.4, Figs. 18.5, 18.6, and 18.7 (Fig. 18.4).

Overall effectiveness. Our method finds a better permutation of test inputs than baselines, i.e., identifying bug-revealing inputs earlier, which significantly improves

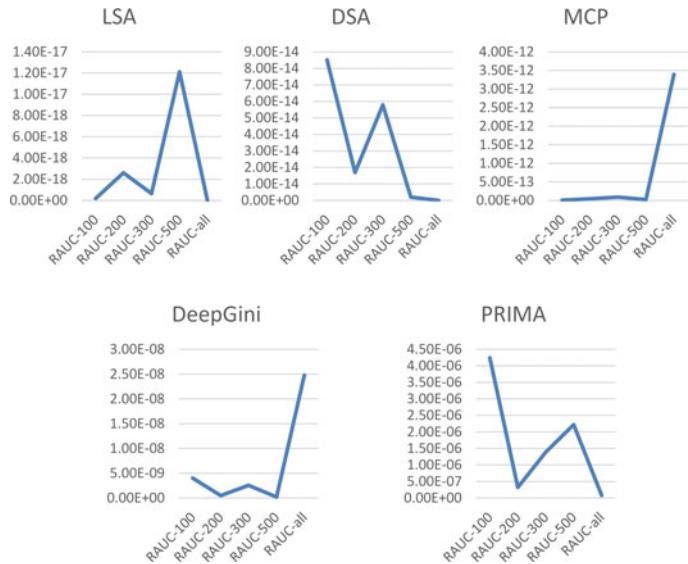


Fig. 18.3 The p-value of T-test for average RAUC between CertPri and each baseline

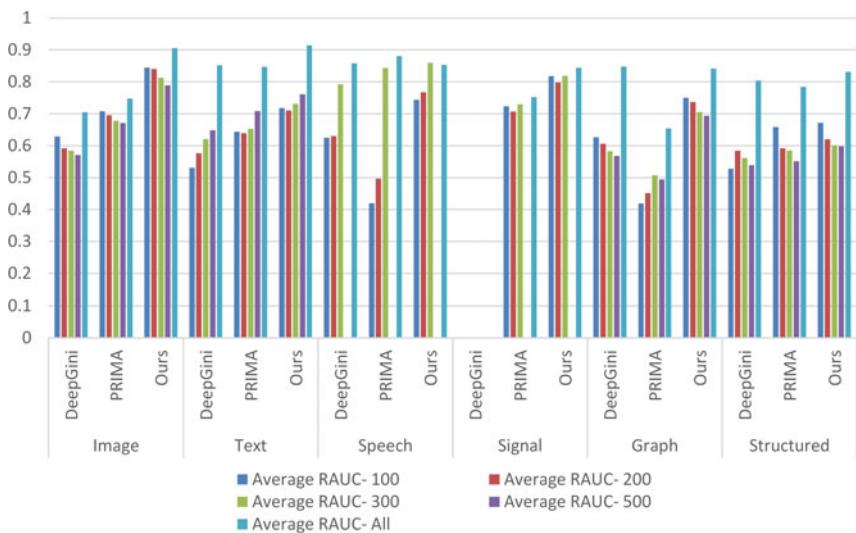


Fig. 18.4 Comparison on various data forms of test inputs across all subjects

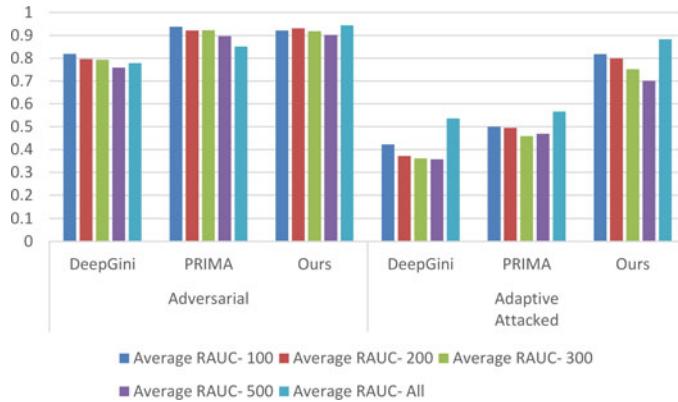


Fig. 18.5 Comparison on various data types of test inputs across the image subjects (ID: 3–8, 11–16, 19–24, 27–32)

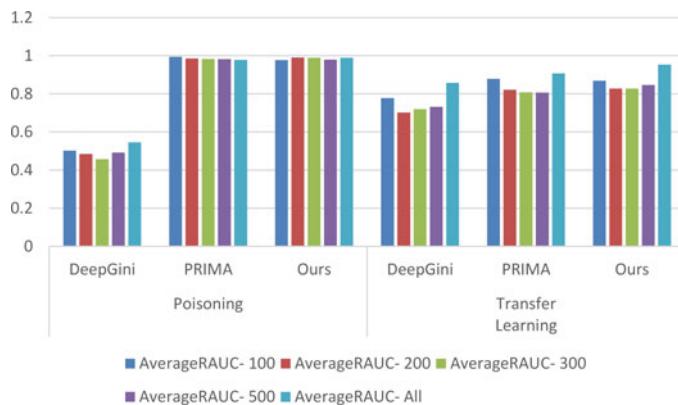


Fig. 18.6 Comparison on various training scenarios across the image subjects (ID: 38–40)

prioritization. For instance, in Fig. 18.2, all average RAUC values of our method are the highest compared with baselines for various tasks. More specifically, the average RAUC value of our method is 1.50 times and 1.42 times that of baselines for classification and regression tasks, respectively. Additionally, our method improves the prioritization effect by 55.39% and 50.41% for classification and regression tasks, respectively. From Fig. 18.3, we can see that the p-values of all RAUC metrics are small enough, which demonstrates that our method significantly outperforms all baselines. The outstanding performance of our method is mainly because it takes formal guarantees into account when identifying bug-revealing inputs while baselines only prioritize empirically. Therefore, our method shows a stable overall prioritization effect without being affected by interfering factors (e.g., various tasks, various data types, etc.).

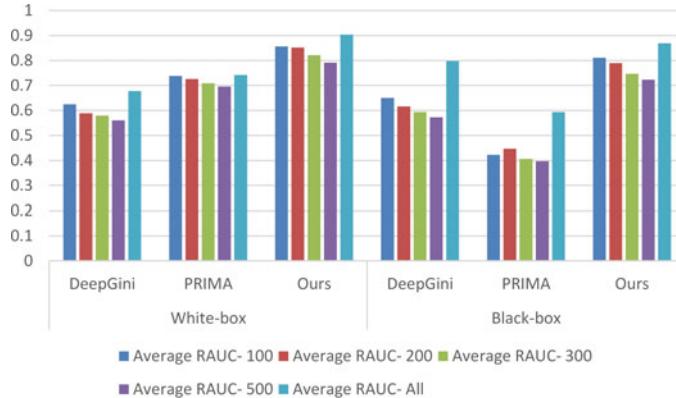


Fig. 18.7 Comparison on various prioritization scenarios across the image subjects (ID: 1–36)

Effectiveness on various data forms of test inputs. Our method outperforms all baselines on all six data forms in terms of almost all RAUC values, especially for unstructured data forms (i.e., image, text, etc.). For instance, in Table 18.4, almost all RAUC values of our method are the highest, except RAUC-all on speech and graph data forms. We investigate their model feature space and find that their decision boundaries are smoother than others, which causes gradient vanishing. Therefore, we extend the sampling radius $\mathcal{R} = 0.05 \max(x)$, which facilitates maximum gradient norm estimation based on GEVT. After radius extension, our method realizes the highest average RAUC-all, improving to 0.9127 and 0.8817 for speech and graph data forms, respectively. Additionally, the average RAUC of our method is 1.13~1.30 times that of baselines for unstructured data forms, but only 1.07 times for structured data. We speculate that the gradient vanishes during backpropagation due to the sparse coding of structured data. Gradient vanishing is beyond the scope of this chapter, but it can be improved by batch normalization [53] and non-saturating activation function [33].

Effectiveness on various data types of test inputs. Our method largely outperforms all baselines against adaptive attacks in terms of all average RAUC values, while approaching the SOTA baseline (i.e., PRIMA) against adversarial attacks. For instance, in Fig. 18.5, the average RAUC values of our method against adaptive attacks range from 0.7006 to 0.8827 with average improvements of 64.73%~114.85% compared with DeepGini and 49.55%~64.11% compared with PRIMA, respectively. This is mainly because we provide a formal guarantee of movement costs in feature space, which cannot be an objective of adaptive attacks. Besides, the average RAUC-100 and RAUC-300 gaps between PRIMA and our method are both less than 0.02, which is acceptable and does not hinder its practical application in various data types.

Effectiveness on various training scenarios. Our method outperforms DeepGini and shows competitive performance with PRIMA in both training scenarios.

For instance, in Fig. 18.6, the average RAUC values of our method range from 0.9769 to 0.9908 with average improvements of 81.20%~116.56% compared with DeepGini in the poisoning scenario, and range from 0.8274 to 0.9532 with average improvements of 11.10%~17.92% compared with DeepGini in transfer learning. Besides, the average RAUC-100 and RAUC-500 gaps between PRIMA and our method are both less than 0.02. We speculate that the purity assumption of DeepGini is not tenable in the two training scenarios [10], whereas our method and PRIMA facilitate their prioritization based on the movement cost and mutation perspectives, respectively.

Effectiveness on various prioritization scenarios. Our method largely outperforms all baselines in terms of all RAUC values in both prioritization scenarios, which is beneficial for identifying bug-revealing inputs in software engineering testing with privacy requirements. For instance, in Fig. 18.7, the average RAUC values of our method range from 0.7909 to 0.9039 with average improvements of 13.66%~44.65% compared with baselines in the white-box scenario, and range from 0.7227 to 0.8688 with average improvements of 8.94%~92.12% compared with baselines in the black-box scenario. The outstanding performance of our method is mainly because it leverages the Weibull distribution to determine the exact maximum gradient norm in the white-box scenario, and adopts gradient estimation to satisfy the black-box scenario.

Answer to RQ1: our method outperforms baselines in two aspects in terms of effectiveness: (1) *overall*-it significantly improves 81.84%, 47.48%, and 18.65% RAUC values on average compared with surprise-based, confidence-based, and mutation-based baselines, respectively; (2) *specific*-it improves 20.66%, 43.39%, 28.96%, and 38.88% RAUC values on average compared with baselines (i.e., DeepGini and PRIMA) for various data forms, data types, training scenarios, and prioritization scenarios, respectively.

18.6.2 Efficiency (RQ2)

How *efficient* is our method in prioritizing test inputs?

When answering this question, we refer to the prioritization time costs in the white-box scenarios with image data form (i.e., ID: 1–36). The evaluation results are shown in Fig. 18.8, where the time cost of PRIMA includes input mutation, model mutation, and ranking model training. Here we have the following observation.

Prioritization efficiency. Our method prioritizes test inputs more efficiently than mutation-based methods and is competitive with confidence-based methods, which meets the rapidity requirements of software engineering testing. For instance, in Fig. 18.8, the efficiency of our method is 41.17 times and 52.86 times that of DSA and PRIMA, respectively. This is mainly because our method only needs to perform gradient computation based on backpropagation and extreme value estimation, both of which are lightweight operations. Besides, in Fig. 18.8, the time cost of our

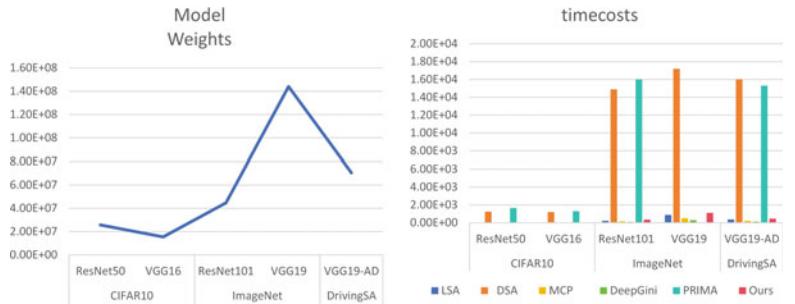


Fig. 18.8 Efficiency comparison across the image subjects (ID: 1–36), measured by “#seconds/1,000 Inputs”

method is 1.25 times and 2.94 times more than that of MCP and DeepGini on average, respectively. The reason is that our method involves iterative operations in the extreme value estimation, which increases time costs. Note that our method adopts a double-minimum strategy to ensure its effectiveness, which leaves room for efficiency improvements. We can slightly reduce the N_b , N_{rsb} , \mathcal{R} values in Algorithm 18.1 to further improve its efficiency without loss of effectiveness.

Answer to RQ2: our method is more efficient in prioritization speed—it prioritizes test inputs with an average speedup of 51.86 times compared with SOTA method (i.e., PRIMA).

18.6.3 Robustness (RQ3)

How *robust* is our method against adaptive attacks based on its certifiability?

When reporting the results, we focus on the following aspects: the robustness against adaptive attacks and the utility of robustness. The evaluation results are shown in Figs. 18.9 and 18.10.

Implementation details for robustness evaluation. (1) To measure our method’s robustness, we refer to the variation of RAUC values between original test inputs (ID: 1, 9, 17, 25) and adaptive attacked test inputs (ID: 6–8, 14–16, 22–24, 30–32), i.e., RobR, as shown in Fig. 18.9. (2) To demonstrate the robustness utility, taking adaptive attacks on the ImageNet dataset as an example (ID: 22–24), we combine our method with each baseline to show the promotion effect of our method on baselines, as shown in Fig. 18.10, where x -axis represents the percentage of our method components added to baselines.

Prioritization robustness. In all cases, our method always performs the most robust prioritization results against various adaptive attacks, which facilitates the

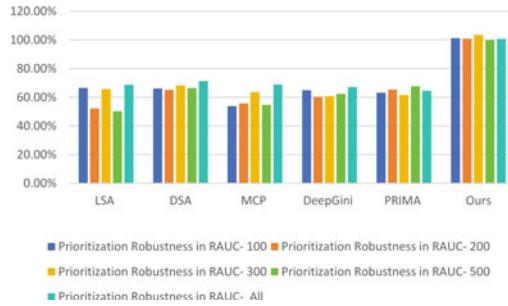


Fig. 18.9 Robustness comparison across the image subjects (ID: 1, 6–9, 14–17, 22–25, 30–32), measured by RobR

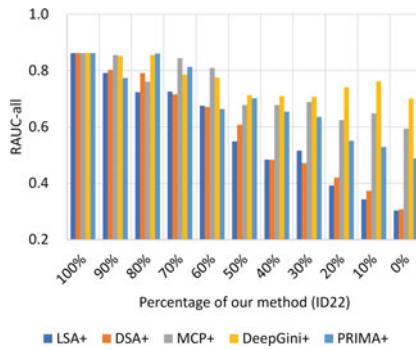


Fig. 18.10 The promotion results of our method for baselines through combining it with each baseline

stable identification of bug-revealing inputs. For instance, in Fig. 18.9 (take ID22 as an example), the average RobR values of our method against adaptive attacks range from 99.93% to 103.60%, which is 1.50~1.71 times that of baselines. The outstanding performance of our method is mainly because it simplifies the prioritization task into a lower bound measure of the movement cost, which has been formally guaranteed in Sect. 18.4.2.

Utility of robustness. Our method is not only immune to various adaptive attacks, but also facilitates the robustness of baselines through weighted combinations. For instance, in Fig. 18.10, all curves show an upward trend after combining with our method and are always higher than their initial value without our method. Furthermore, we speculate that for an empirical prioritization method that outperforms our method in general, it can be combined with our method to improve robustness against adaptive attacks.

Answer to RQ3: our method largely outperforms baselines in terms of robust prioritization with average robustness improvements of 41.37%~98.91%. Besides, its robustness can be leveraged to facilitate other methods.

18.6.4 Generalizability (*RQ4*)

How generic is our method in prioritizing test inputs?

When reporting the generalizability, we focus on the ranking of RAUC values for different methods in each subject. The evaluation results are illustrated as radar charts in Table 18.1.

Implementation details for generalizability evaluation. (1) Calculate the GenRew value of each dimension separately. Take the task dimension as an example, which includes classification and regression. First, we select all subjects belonging to the classification task and calculate GenRew according to Eq. (18.10), denoted as GR_c . Then, we select all subjects belonging to the regression task and compute GenRew, denoted as GR_r . Finally, we compute the average of GR_c and GR_r as the task-dimensional GenRew. (2) Repeat the above operations for the remaining five dimensions (i.e., data form/type, structure, training/prioritization scenarios).

Prioritization generalizability. Our method always outperforms all baselines against various dimensions in terms of all average GenRew values. For instance, in Table 18.1, the area of our method covers all baselines. More specifically, the average GenRew values of our method range from 0.9040 to 0.9813 with average improvements of 32.81%~238.54% compared with baselines. This is mainly because our method's calculation only involves gradient derivation based on backpropagation, which is easy to implement for any DNN. Thus, our method can be generalized to various dimensions.

Answer to RQ4: our method is more generic than baselines in all six dimensions with average generalizability improvements of 32.81%~238.54%.

18.7 Conclusions

We propose a certifiable prioritization method for bug-revealing test input identification earlier. Our method aims to efficiently address the labeling-cost problem in DNN testing and build trustworthy deep learning systems. Our method provides a new perspective on prioritization, which reduces the problem of measuring misbehavior probability to the problem of measuring the difficulty of movement in feature space. Based on this view, we give formal guarantees about lower bounds γ_L on movement cost and compute γ_L value based on GEVT. The priority of each test input is determined in ascending order of γ_L value. Furthermore, we generalize our method in black-box scenarios by gradient estimation. Our method is compared with baseline on various tasks, data forms, data types, model structures, training, and prioritization scenarios. The results show that our method outperforms baselines in terms of effectiveness, efficiency, robustness, and generalizability.

Table 18.1 Generalizability comparison for six dimensions across all subjects, measured by Gen-Rew

Dimensions	Methods	Average RAUC-				
		100	200	300	500	All
Tasks	LSA	0.4701	0.3746	0.3884	0.3557	0.4564
	DSA	0.2701	0.2795	0.2809	0.2780	0.2644
	MCP	0.2941	0.3366	0.3360	0.3100	0.3458
	DeepGini	0.4208	0.4129	0.4008	0.4091	0.4267
	PRIMA	0.7824	0.6915	0.6927	0.6285	0.7166
	Ours	0.9595	0.9068	0.9031	0.8255	0.9870
Data form	LSA	0.3676	0.2726	0.2844	0.2542	0.3731
	DSA	0.2602	0.3031	0.2986	0.2789	0.2465
	MCP	0.4871	0.4993	0.5005	0.3547	0.4520
	DeepGini	0.5764	0.5937	0.5406	0.4788	0.6580
	PRIMA	0.7147	0.6420	0.7059	0.4750	0.7247
	Ours	0.9681	0.9421	0.9228	0.6613	0.9200
Data type	LSA	0.3944	0.2787	0.3106	0.3120	0.3646
	DSA	0.3769	0.4009	0.4056	0.4028	0.3641
	MCP	0.3854	0.4771	0.4676	0.4252	0.4991
	DeepGini	0.6421	0.6213	0.5995	0.6213	0.6484
	PRIMA	0.6979	0.6597	0.6623	0.6461	0.5889
	Ours	0.9546	0.9512	0.9433	0.9190	0.9863
Model structure	LSA	0.3465	0.2304	0.2476	0.2961	0.3432
	DSA	0.2796	0.3187	0.3119	0.3136	0.2709
	MCP	0.5136	0.5169	0.5545	0.3984	0.4907
	DeepGini	0.5942	0.6174	0.5478	0.5616	0.7093
	PRIMA	0.7195	0.6221	0.6512	0.5667	0.6621
	Ours	0.9528	0.9193	0.9117	0.8070	0.9301
Training scenario	LSA	0.2872	0.2066	0.2152	0.2173	0.3440
	DSA	0.4475	0.4051	0.3842	0.3171	0.4599
	MCP	0.3635	0.4238	0.4778	0.4181	0.3798
	DeepGini	0.5936	0.4871	0.4469	0.5609	0.5320
	PRIMA	0.8572	0.6885	0.6908	0.7025	0.7699
	Ours	0.9190	0.8261	0.8223	0.7895	0.9825
Prioritization scenario	LSA	0.2874	0.2293	0.2428	0.2459	0.2741
	DSA	0.2678	0.2770	0.2783	0.2756	0.2622
	MCP	0.4276	0.4661	0.4656	0.4787	0.5463
	DeepGini	0.6522	0.6444	0.6326	0.6407	0.6313
	PRIMA	0.6628	0.6356	0.6391	0.5846	0.5687
	Ours	0.9744	0.9698	0.9639	0.9467	0.9896

References

1. Ma, L., Juefei-Xu, F., Zhang, F., Sun, J., Xue, M., Li, B., Chen, C., Su, T., Li, L., Liu, Y.: Deepgauge: Multi-granularity testing criteria for deep learning systems. In: 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 120–131. ACM, Montpellier (2018)
2. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: automated whitebox testing of deep learning systems. *Commun. ACM* **62**(11), 137–145 (2019)
3. Wicker, M., Huang, X., Kwiatkowska, M.: Feature-guided black-box safety testing of deep neural networks. In: Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference. TACAS 2018, vol. 10805, pp. 408–426. Springer, Thessaloniki (2018)
4. Zhang, K., Zhang, Y., Zhang, L., Gao, H., Yan, R., Yan, J.: Neuron activation frequency based test case prioritization. In: International Symposium on Theoretical Aspects of Software Engineering, pp. 81–88. IEEE, Hangzhou (2020)
5. Byun, T., Sharma, V., Vijayakumar, A., Rayadurgam, S., Cofer, D.D.: Input prioritization for testing neural networks. In: IEEE International Conference On Artificial Intelligence Testing. AITest 2019, pp. 63–70. IEEE, Newark (2019)
6. Kim, J., Feldt, R., Yoo, S.: Guiding deep learning system testing using surprise adequacy. In: 41st IEEE/ACM International Conference on Software Engineering, pp. 1039–1049. IEEE/ACM, Montreal, QC, Canada (2019)
7. Feng, Y., Shi, Q., Gao, X., Wan, J., Fang, C., Chen, Z.: Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 177–188. ACM, Virtual Event, USA (2020)
8. Shen, W., Li, Y., Chen, L., Han, Y., Zhou, Y., Xu, B.: Multiple-boundary clustering and prioritization to promote neural network retraining. In: 35th IEEE/ACM International Conference on Automated Software Engineering, pp. 410–422. IEEE, Melbourne, Australia (2020)
9. Zhang, L., Sun, X., Li, Y., Zhang, Z.: A noise-sensitivity-analysis-based test prioritization technique for deep neural networks. *CoRR* pp. 1–8 (2019)
10. Wang, Z., You, H., Chen, J., Zhang, Y., Dong, X., Zhang, W.: Prioritizing test inputs for deep neural networks via mutation analysis. In: 43rd IEEE/ACM International Conference on Software Engineering, pp. 397–409. IEEE, Madrid, Spain (2021)
11. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, pp. 1–11. OpenReview.net, San Diego, CA, USA (2014)
12. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 9185–9193. Computer Vision Foundation/IEEE Computer Society, Salt Lake City, UT, USA (2018)
13. Chen, J., Zheng, H., Xiong, H., Shen, S., Su, M.: Mag-gan: Massive attack generator via gan. *Inf. Sci.* **536**(1), 67–90 (2020)
14. Moosavi-Dezfooli, S.M., Fawzi, A., Frossard, P.: Deepfool: a simple and accurate method to fool deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 2574–2582. IEEE Computer Society, Las Vegas, NV, USA (2016)
15. Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. *J. Big Data* **6**(1), 1–48 (2019)
16. Sun, B., Tsai, N.H., Liu, F., Yu, R., Su, H.: Adversarial defense by stratified convolutional sparse coding. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 11447–11456. Computer Vision Foundation/IEEE, Long Beach, CA, USA (2019)
17. Mustafa, A., Khan, S., Hayat, M., Goecke, R., Shen, J., Shao, L.: Adversarial defense by restricting the hidden space of deep neural networks. In: IEEE International Conference on Computer Vision, pp. 3384–3393. IEEE, Seoul, South Korea (2019)
18. Rakin, A.S., Zhezhi, H., Deliang, F.: Parametric noise injection: trainable randomness to improve deep neural network robustness against adversarial attack. In: IEEE Conference on

- Computer Vision and Pattern Recognition, pp. 588–597. Computer Vision Foundation/IEEE, Long Beach, CA, USA (2019)
- 19. Li, Y., Hua, J., Wang, H., Chen, C., Liu, Y.: Deeppayload: Black-box backdoor attack on deep learning models through neural payload injection. In: 43rd IEEE/ACM International Conference on Software Engineering (ICSE 2021), pp. 263–274. IEEE, Madrid, Spain (2021)
 - 20. Alshemali, B., Kalita, J.: Improving the reliability of deep neural networks in nlp: a review. *Knowl.-Based Syst.* **191**(1), 1–19 (2020)
 - 21. Bellamy, R.K.E., Dey, K., Hind, M., Hoffman, S.C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K.N., Richards, J.T., Saha, D., Sattigeri, P., Singh, M., Varshney, K.R., Zhang, Y.: AI fairness 360: an extensible toolkit for detecting and mitigating algorithmic bias. *IBM J. Res. Dev.* **63**(4/5), 1–15 (2019)
 - 22. Zhang, Z., Cui, P., Zhu, W.: Deep learning on graphs: A survey. *IEEE Trans. Knowl. Data Eng.* **34**(1), 249–270 (2020)
 - 23. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
 - 24. Zheng, H., Chen, J., Du, H., Zhu, W., Ji, S., Zhang, X.: Grip-gan: An attack-free defense through general robust inverse perturbation. *IEEE Trans. Depend. Secure Comput.* 1–18 (2021)
 - 25. Paulavičius, R., Žilinskas, J.: Analysis of different norms and corresponding lipschitz constants for global optimization. *Technol. Econ. Dev. Econ.* **12**(4), 301–306 (2006)
 - 26. Gnedenko, B.: Sur la distribution limite du terme maximum d'une serie aleatoire. *Ann. Math.* **44**(3), 423–453 (1943)
 - 27. Ma, W., Papadakis, M., Tsakmalis, A., Cordy, M., Traon, Y.L.: Test selection for deep learning systems. *ACM Trans. Softw. Engin. Methodol.* (TOSEM) **30**(2), 1–22 (2021)
 - 28. Xie, X., Yin, P., Chen, S.: Boosting the revealing of detected violations in deep learning testing: a diversity-guided method. In: 37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10–14, 2022, pp. 17:1–17:13. ACM (2022)
 - 29. Xie, X., Li, T., Wang, J., Ma, L., Guo, Q., Juefei-Xu, F., Liu, Y.: NPC: neuron path coverage via characterizing decision logic of deep neural networks. *ACM Trans. Softw. Eng. Methodol.* **31**(3), 47:1–47:27 (2022)
 - 30. Lou, Y., Hao, D., Zhang, L.: Mutation-based test-case prioritization in software evolution. In: 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 46–57. IEEE Computer Society, Gaithersburg, MD, USA (2015)
 - 31. Shin, D., Yoo, S., Papadakis, M., Bae, D.H.: Empirical evaluation of mutation-based test case prioritization techniques. *Softw. Testing, Verific. Reliab.* **29**(1–2), 1–2 (2019)
 - 32. Denker, J.S., LeCun, Y.: Transforming neural-net output levels to probability distributions. In: Advances in Neural Information Processing Systems 3 (NIPS 1990), pp. 853–859. Morgan Kaufmann, Denver, Colorado, USA (1990)
 - 33. Eckle, K., Schmidt-Hieber, J.: A comparison of deep networks with relu activation function and linear spline-type methods. *Neural Netw.* **110**, 232–242 (2019)
 - 34. Weng, T., Zhang, H., Chen, P., Yi, J., Su, D., Gao, Y., Hsieh, C., Daniel, L.: Evaluating the robustness of neural networks: An extreme value theory approach. In: 6th International Conference on Learning Representations (ICLR 2018), pp. 1–18. OpenReview.net, Vancouver, BC, Canada (2018)
 - 35. Chen, J., Zheng, H., Shangguan, W., Liu, L., Ji, S.: Act-detector: Adaptive channel transformation-based light-weighted detector for adversarial attacks. *Inf. Sci.* **564**, 163–192 (2021)
 - 36. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W., Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1**(4), 541–551 (1989)
 - 37. Van der Maaten, L., Hinton, G.: Visualizing data using t-sne. *J. Mach. Learn. Res.* **9**(11), 2579–2605 (2008)
 - 38. Amari, S.I.: Backpropagation and stochastic gradient descent method. *Neurocomputing* **5**(4), 185–196 (1993)

39. Chen, P., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.: ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, pp. 15–26. ACM, Dallas, TX, USA (2017)
40. Krizhevsky, A.: Learning multiple layers of features from tiny images. University of Toronto, Technical report, Computer Science Department (2009)
41. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. *Int. J. Comput. Vis. (IJCV)* **115**(3), 211–252 (2015)
42. Deng, Y., Zheng, J.X., Zhang, T., Chen, C., Lou, G., Kim, M.: An analysis of adversarial attacks and defenses on autonomous driving models. In: IEEE International Conference on Pervasive Computing and Communications (PerCom 2020), pp. 1–10. IEEE, Austin, TX, USA (2020)
43. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. ArXiv Preprint pp. 1–6 (2017)
44. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pp. 142–150. Association for Computational Linguistics, Portland, Oregon, USA (2011)
45. O’Shea, T.J., West, N.: Radio machine learning dataset generation with gnu radio. In: Proceedings of the GNU Radio Conference, pp. 1–10 (2016)
46. McCallum, A., Nigam, K., Rennie, J., Seymore, K.: Automating the construction of internet portals with machine learning. *Inf. Retr.* **3**(2), 127–163 (2000)
47. Kohavi, R.: Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, pp. 202–207. AAAI Press, Menlo Park, CA (1996)
48. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations (ICLR 2017), pp. 1–14. OpenReview.net, Toulon, France (2017)
49. Carlini, N., Wagner, D.: Towards evaluating the robustness of neural networks. In: IEEE Symposium on Security and Privacy (SP 2017), pp. 39–57. IEEE Computer Society, San Jose, CA, USA (2017)
50. Chen, J., Zheng, H., Xiong, H., Chen, R., Du, T., Hong, Z., Ji, S.: Finefool: A novel DNN object contour attack on image recognition based on the attention perturbation adversarial technique. *Comput. Secur.* **104**, 102220 (2021)
51. Hosseini, H., Poovendran, R.: Semantic adversarial examples. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops. CVPR Workshops 2018, pp. 1614–1619. Computer Vision Foundation/IEEE Computer Society, Salt Lake City, UT, USA (2018)
52. Chen, J., Zhang, L., Zheng, H., Wang, X., Ming, Z.: Deepoison: feature transfer based stealthy poisoning attack for dnns. *IEEE Trans. Circuits Syst. II Express Briefs* **68**(7), 2618–2622 (2021)
53. Ioffe, S., Szegedy, C.: Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd International Conference on Machine Learning (ICML 2015). vol. 37, pp. 448–456. JMLR.org, Lille, France (2015)

Chapter 19

Interpretable White-Box Fairness Testing Through Biased Neuron Identification



19.1 Introduction

Deep neural networks (DNNs) [1] have been increasingly adopted in many fields. However, one of the crucial factors hindering DNNs from further serving applications with social impact is the unintended individual discrimination [2–4]. Individual discrimination occurs when an instance differs from other instances only in sensitive attributes (e.g., gender, race), but the predictions obtained from a given DNN are different [5]. Taking the example of gender discrimination in wage forecasting, the two cases are identical except for the gender attribute, male's annual income predicted by the DNN is often higher than female's [6]. Therefore, it is important for stakeholders to identify fairness violations and reduce discrimination in DNNs in order to responsibly implement fair and reliable deep learning systems in many sensitive scenarios [7–11].

Much effort has been put into uncovering fairness violations [12–14]. The most common method is fairness testing [5, 15, 16] which solves this problem by generating as many instances as possible. Initially, fairness testing is aimed at discovering and reducing discrimination in traditional machine learning (ML) using low-dimensional linear models. However, there are several problems with these methods. First, most of them (e.g., FairAware [15], BlackFT [5], and FlipTest [16]) cannot handle DNNs with high-dimensional non-linear structures. Then, though some of them (e.g., Themis [17], SymbGen [18], and Aequitas [19]) can be applied to test DNNs, they are still challenged by the high time cost and numerous duplicate instances. Recently, several methods have been specifically developed for DNNs, such as ADF [20] and EIDIG [21]. These methods make progress in effectiveness and efficiency through gradient guidance, but they still suffer from the following problems.

First, these methods can hardly be generalized to unstructured data. As we know, DNNs are originally designed to process unstructured data (e.g., image, text, speech, etc.), but most existing fairness testing methods [17–19] cannot be applied to these data. This is mainly due to the fact that these methods are unable to determine which

features are associated with sensitive attributes and to modify them appropriately. As a result, these testing methods cannot be widely used for DNNs until the data generalization problem has been solved.

Second, the generation effectiveness of these methods is threatened by gradient loss. Although these methods utilize gradient-inducing strategies to improve generation efficiency, they may fail to generate instances due to gradient loss. Moreover, when the gradient is small, the generated instances are very similar. However, the purpose of fairness testing is not only to generate a large number of instances, but also to generate a wide variety of instances.

Third, almost all existing methods have poor interpretability. They focus only on generating a large number of instances but fail to explain how biased decisions are made. In summary, the current challenges for fairness testing lie in the lack of generalizability of the data, the effectiveness of generation, and the interpretation of discrimination.

To overcome the above challenges, our design goals are as follows: (1) we intend to uncover and quantitatively interpret DNNs' discrimination; (2) then, we plan to apply this interpretation results to guide fairness testing; (3) furthermore, we want to generalize our testing method to unstructured data. Imagine if the decision outcome of a DNN is affected by certain neurons, since the decision is determined by a non-linear combination of the activation states of each neuron. Next, we give two pairs of examples to observe the activation states of neurons in the DNN hidden layer. Surprisingly, we find that the activation state follows such a pattern, i.e., neurons with drastically varying activation values are overlapping for different instance pairs. It is observed that the removal of these duplicate neurons decreases the DNN recognition rate. Therefore, it can be inferred that these neurons lead to DNN recognition. Next, we tried to quantitatively explain DNN recognition by calculating the activation difference (ActDiff) value of the neurons.

Based on the interpretation of the results, we further designed a test method to optimize gradient induction. First, we identify the key neurons responsible for discrimination, called bias neurons. Then, we search for discriminatory instances with the optimization object of increasing the ActDiff values of biased neurons. Because the optimization from the biased neuron shortens the derivation path, it reduces the probability of the gradient vanishing and time cost. In addition, by dynamically combining the biased neurons, more diverse instances can be generated. Finally, the interpretation results can be used to optimize the gradient direction, thus improving the generation efficiency.

We leverage adversarial attacks [22–24] to determine which features are related to sensitive attributes, and make appropriate modifications to these features. The adversarial attack is originally to test the DNNs' security, e.g., slight modifications to some image pixels will cause the predicted label to flip [25–28]. Taking the gender attribute of a face image as an example, we train a classifier with “male” and “female” labels, then adding the perturbation to the face image until its predicted gender label flips. Based on this generalization framework, we can change the sensitive properties of any data and thus generalize our methods to any data type.

In summary, we first implement a quantitative interpretation of the discriminant using neural-based analysis, then use the interpretation results to optimize instance generation, and finally design a generalization framework for sensitive attribute modification. The main contributions are as follows:

- Through the neuron activation analysis, we quantitatively interpret DNNs' discrimination, which provides a new perspective for measuring discrimination and guides DNNs' fairness testing.
- Based on the interpretation results, we design a novel method for DNNs' discriminatory instance generation, which significantly outperforms previous works in terms of effectiveness.
- Inspired by adversarial attacks, we design a generalization framework to modify sensitive attributes of unstructured data, which generalizes our method to unstructured data.
- We publish our method as a self-contained open-source toolkit online.

19.2 Related Works

Fairness Testing. Based on the software engineering point of view, several works on testing the fairness of traditional ML models are proposed [4, 17–19, 29, 30]. To uncover their fairness violations, Galhotra et al. [17] firstly proposed Themis, a fairness testing method for software, which measures the discrimination in software through counting the frequency of IDIs in the input space. However, its efficiency for IDIs generation is unsatisfactory. To improve the generation speed of Themis, Udeshi et al. [19] proposed a faster generation algorithm, Aequitas, which uncovers fairness violations by probabilistic search over the input space. Aequitas adopts a two-phase operation in which the IDIs generated globally are used as seeds for the local generation. However, Aequitas uses a global sampling distribution for all the inputs, which leads to the limitation that it can only search in narrow input space and easily falls into the local optimum. Thus, Aequitas's IDIs lack diversity. To further improve the instance diversity, Agarwal et al. [18] designed a new testing method, SymbGen, which combines the symbolic execution along with the local interpretation for the generation of effective instances. SymbGen first constructs the complex model's local explainer and then searches for IDIs based on the fitted decision boundary. Therefore, its instance effectiveness almost depends on the performance of the explainer.

The methods mentioned above mainly deal with traditional ML models, which cannot directly be applied to deal with DNNs. Recently, several methods have been proposed specifically for DNNs. For instance, Zhang et al. [20] first proposed a fairness testing method specifically for DNNs, ADF, which guides the search direction through gradients. The authors proved that the effectiveness and efficiency of IDIs generation for DNNs are greatly improved based on the guidance of gradients. Based on the ADF [20], Zhang et al. [21] designed a framework EIDIG for discovering individual fairness violations, which adopts prior information to accelerate the convergence of iterations. However, there is still a problem of gradient vanishing, which may lead to a local optimum.

Neuron-based DNN Interpretation. Kim et al. [31] first introduced concept activation vectors, which provide an interpretation of a DNN’s internal state (i.e., the activation output in the hidden layer). They viewed the high-dimensional internal state of a DNN as an aid and interpreted which concept is important to the classification result. Inspired by the concept of activation vectors, Du et al. [32] suggested that interpretability can serve as a useful ingredient to diagnose the reasons that lead to algorithmic discrimination. While previous methods studied the activation output of the hidden layer, Liu et al. [33] studied the activation state of individual neurons. They observed that the neuron activation is related to the DNNs’ robustness, and used the abnormal activation of a single neuron to detect backdoor attacks. These methods utilize internal states to explain the classification performance and robustness of DNNs, which inspires us to use them to interpret DNNs’ biased decisions.

19.3 Background

To better understand the problem we are tackling and the methodology we propose in later sections, we first introduce data form, individual discrimination, and our problem definition.

Data Form. Denote $X = \{x_i\}$, $Y = \{y_i\}$ as a normal dataset, and its instance pairs by $\langle X, X' \rangle = \langle x_i, x'_i \rangle$, $i \in \{0, 1, \dots, N - 1\}$. For an instance, we denote its attributes by $A = \{a_i\}$, $i \in \{0, 1, \dots, N_a - 1\}$, where $A_s \subset A$ is a set of sensitive attributes, and $A_{ns} = \{a_i^{ns} | a_i^{ns} \in A, \text{ and } a_i^{ns} \notin A_s\}$ is a set of non-sensitive attributes. Note that sensitive attributes (e.g., gender, race, age, etc.) are usually given in advance according to specific sensitive scenes.

Individual Discrimination. As stated in previous work [5, 15], individual discrimination exists when two valid inputs which differ only in the sensitive attributes but receive a different prediction result from a given DNN, as shown in Fig. 19.1. Such two valid inputs are called individual discriminatory instances (IDIs).

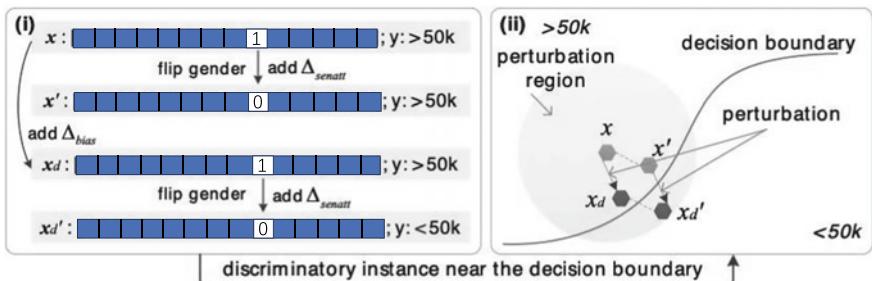


Fig. 19.1 Illustration of discriminatory instance generation on Adult dataset [6]. **(i)** The discriminatory instance generation process. The normal instance pair is $\langle x, x' \rangle$ and the discriminatory instance pair is $\langle x_d, x'_d \rangle$. $x' = x + \Delta_{senatt}$, $x_d = x + \Delta_{bias}$, $x'_d = x + \Delta_{senatt} + \Delta_{bias}$, where Δ_{bias} is the bias perturbation, Δ_{senatt} is the perturbation added to the gender attribute to flip gender. **(ii)** Discrimination exists when the instance's predicted label changes as the gender attribute is flipped, i.e., the instance crosses the decision boundary

Definition 19.1 (*IDI determination*) We denote $\langle X_d, X'_d \rangle = \{ \langle x_{d,i}, x'_{d,i} \rangle \}$ as a set of IDI pairs, which satisfies

$$\begin{aligned} f(x_{d,i}; \Theta) &\neq f(x'_{d,i}; \Theta) \\ \text{s.t. } x_{d,i}[A_s] &\neq x'_{d,i}[A_s], \quad x_{d,i}[A_{ns}] = x'_{d,i}[A_{ns}], \end{aligned} \quad (19.1)$$

where $i \in \{0, 1, \dots, N_d - 1\}$, $x_{d,i}[A_s]$ represents the value of $x_{d,i}$ with respect to attribute A_s . Note that our instances are generative (e.g., maybe the age of a generated instance is 150 years old on Adult dataset), thus we need to clip their attribute values that do not exist in the input domain \mathbb{I} .

19.4 Methodology

An overview of our method is presented in Fig. 19.2. Our method has two parts, i.e., discrimination interpretation and IDI generation based on interpretation results. The explanation of discrimination first uses a neuron-based analysis to explain why discrimination exists. Then, we design a discrimination metric based on the interpretation result, i.e., AUC value, as shown in Fig. 19.2 (i). AUC is the area under AS curve, where the AS curve records the percentage of neurons above the ActDiff threshold. Finally, the AS curve is used to adaptively identify biased neurons useful for IDI generation. During IDI generation, biased neurons are used for global and local generation. The global phase guarantees the diversity of the generated instances, and the local phase guarantees the quantity, as shown in Fig. 19.2 (ii). On the one hand, the global generation uses the normal instance as a seed and stops if an IDI is generated or it times out. On the other hand, the generated IDIs are adopted as seeds of local generation, leading to generate as many IDIs as possible near the seeds. Besides, we implement dynamic combinations of biased neurons to increase diversity, and use the momentum strategy to accelerate IDI generation. In the following, we first quantitatively interpret DNNs' discrimination, then present details of IDI generation based on interpretation results, and finally generalize our method to unstructured data.

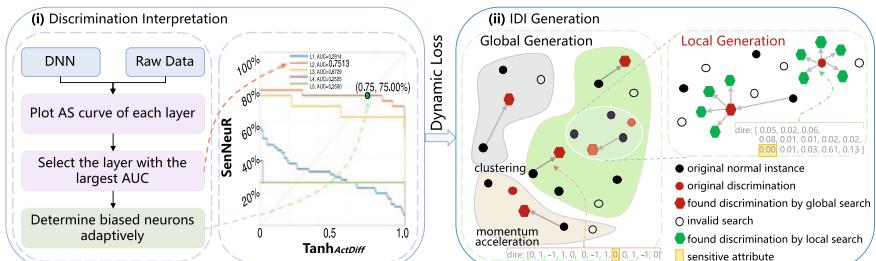


Fig. 19.2 An overview of our method

19.4.1 Quantitative Discrimination Interpretation

First, the discriminability of the DNN is measured by plotting AS curves and calculating AUC values. Then, based on the measurement results, the main neurons that lead to unfair decisions are identified as biased neurons.

19.4.1.1 Discrimination Measurement

The ActDiff is calculated as follows:

$$z_l^k = \frac{1}{N} \sum_{i=0}^{N-1} [\text{abs}(f_l^k(x_i; \Theta) - f_l^k(x'_i; \Theta))], \quad (19.2)$$

where z_l^k is the ActDiff of the k -th neuron in the l -th layer, $l \in \{1, 2, \dots, N_l\}$, N_l is the layer number, N is the number of normal instance pairs $\langle X, X' \rangle = \{\langle x_i, x'_i \rangle\}, i \in \{0, 1, 2, \dots, N-1\}$, $\text{abs}(\cdot)$ returns an absolute value, $f_l^k(x; \Theta)$ returns the activation output of the k -th neuron in the l -th layer, and Θ represents the model weights.

Based on Eq. (19.2), we plot AS curve and compute AUC value. First, the ActDiff of each neuron is calculated and normalized using the hyperbolic tangent function $\text{rmTanh}(\cdot)$, as shown in Fig. 19.3 (i). “L1” means the 1-st hidden layer of a DNN with 64 neurons. Then, multiple ActDiff thresholds are set at equal time intervals and the proportion of neurons exceeding the ActDiff threshold is calculated and recorded as the sensitive neuron ratio (SenNeuR). Finally, the AS curve is plotted based on the SenNeuR under different ActDiff thresholds, and the area under the AS curve is calculated as the AUC, as shown in Fig. 19.3 (ii), where the x-axis is the ActDiff value normalized by Tanh function, the y-axis is SenNeuR. Repeating such an operation for each layer, we can intuitively observe the discrimination in each layer and find the most biased layer with the largest AUC value. As shown in Fig. 19.2 (i), the 2-nd layer “L2” is selected as the most biased layer with AUC=0.7513.

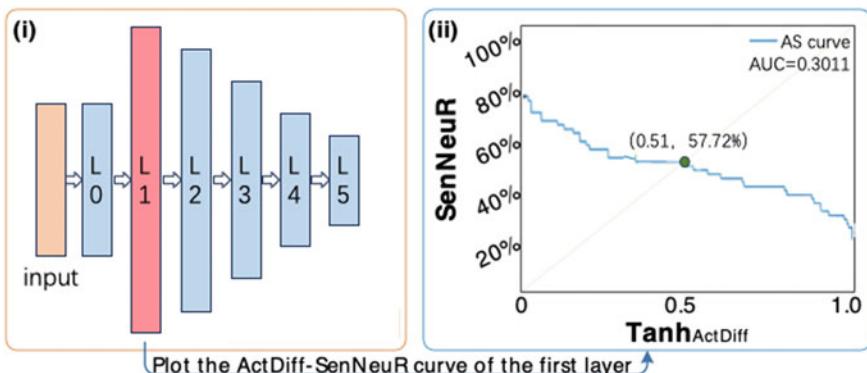


Fig. 19.3 Illustration of the neuron-based discrimination interpretation. Dataset: Adult [6]; dimension: 13; FCN-based DNN structure: [13, 64, 32, 16, 8, 4, 1]

More specific operations on AS curve drawing and AUC calculation are shown in Algorithm 19.1. First, we compute the average ActDiff values of each neuron z_l^k at line 1. In the loop from lines 7 to 9, for each neural layer, we get SenNeuR for plotting the AS curve. Then, we compute AUC value by integration at line 11.

Algorithm 19.1: AS curve drawing and AUC calculation.

Input: The activation output $f_l^k(x; \Theta)$, ActDiff threshold interval $step_interval = 0.005$, instance pairs $\langle X, X' \rangle$.

Output: AS curve and AUC value of each layer.

```

1 Calculate the average ActDiff of each neuron:
   
$$z_l^k = \frac{1}{N} \sum_{i=0}^{N-1} [\text{abs}(f_l^k(x_i; \Theta) - f_l^k(x'_i; \Theta))]$$

2 For  $l = 1 : N_l$ 
3    $z_l = \text{Tanh}(z_l)$ 
4    $max\_z = \max(z_l)$ 
5    $x_{tmp} = 0 : step\_interval : max\_z$ 
6    $y_{tmp} = []$ 
7   For  $count = 1 : \text{length}(x_{tmp})$ 
8      $y_{tmp} = [y_{tmp}, \text{length}(\text{find}(z_l > x_{tmp}[count]))]$ 
9   End For
10   $y_{tmp} = y_{tmp}/\text{length}(z_l) \times 100\%$ 
11   $area = \sum_{count=1}^{\text{length}(x_{tmp})} (y_{tmp}[count] \times step\_interval)$ 
12  Plot the AS curve based on  $(x_{tmp}, y_{tmp})$ .
    Save  $area$  as the AUC of the  $l$ -th layer.
13 End For

```

19.4.1.2 Biased Neuron Identification

To recognize adaptively biased neurons, the layer with the most bias is selected. Neurons with larger values of z_l^k are more responsive to changes in sensitive attributes and thus produce more discrimination. We define biased neuron as follows.

Definition 19.2 (*Biased neuron*) For a given discrimination threshold T_d of the most biased layer, the biased neurons satisfy the condition $z^k > T_d$. z^k is the average ActDiff normalized by $\text{Tanh}(\cdot)$ of the k -th neuron in the most biased layer, $k \in \{1, 2, \dots, N^k\}$, $T_d \in (0, 1)$.

Based on the Definition 19.2, we know that once T_d is determined, biased neurons can be found. A strategy for adaptively determining T_d is given here. We draw a line $y = x$ that intersects the AS curve. The x-axis's value of this intersection is T_d . As shown in Fig. 19.3 (ii), the intersection is the point (0.51, 57.72%) and $T_d = 0.51$. After determining T_d , we record these biased neurons and save their position p , where p is a vector with N^k elements.

This strategy is very effective in practice because the recognition is usually concentrated on a few specific neurons, resulting in normally distributed ActDiff's frequency plots.

Algorithm 19.2: Global generation guided by biased neurons.

Input: Normal instance $X = \{x_i\}$, initial set $\Omega_g = \emptyset$, $X_c = \text{KMeans}(X, N_c)$, $c \in \{1, 2, \dots, N_c\}$, the number of seeds for global generation num_g , the maximum number of iterations for each seed max_iter_g , the perturbation size of each iteration $step_size_g$, the decay rate of momentum μ_g , the step size for random disturbance r_step_g .
Output: A set of IDI pairs found globally Ω_g .

```

1 For  $i = 0 : \text{INT}(num_g/N_c) - 1$ 
2   For  $c = 1 : N_c$ 
3     Select seed  $x$  from  $X_c$ ,  $g_0 = g'_0 = 0$ .
4     For  $t = 0 : max\_iter_g$ 
5       If ( $\text{mod}(step\_size_g, r\_step_g) == 0$ ) Then
6          $r = \text{Rand}_{(0,1)}(p_r)$ 
7       End If
8       Create  $\langle x, x' \rangle$  s.t.  $x[A_s] \neq x'[A_s]$ ,  $x[A_{ns}] = x'[A_{ns}]$ .
9       If ( $f(x; \Theta) \neq f(x'; \Theta)$ ) Then
10         $\Omega_g = \Omega_g \cup \langle x, x' \rangle$ 
11        break
12      End If
13       $g_{t+1} = \mu_g \times g_t + \nabla_x J_{dl}(x; \Theta)$ 
14       $g'_{t+1} = \mu_g \times g'_t + \nabla_{x'} J_{dl}(x'; \Theta)$ 
15       $dire = \text{sign}(g_{t+1} + g'_{t+1})$ 
16       $dire[A_s] = 0$ 
17       $x = x + dire \times step\_size_g$ 
18       $x = \text{Clip}(x, \mathbb{I})$ 
19    End For
20  End For
21 End For
```

19.4.2 Interpretation-Based IDI Generation

Our method generates IDIs in two phases, i.e., a global generation phase and a local generation phase. The goal of the global generation phase is to obtain a diverse set of IDIs. The diversity of IDIs in the global generation phase is crucial because these instances will be the seeds for the local generation phase. Instead, to guarantee the IDIs' quantity, the local phase aims to search for as many IDIs as possible near the seeds.

Algorithm 19.3: Local generation guided by biased neurons.

Input: IDI pairs $\Omega_g = \{< x_{d,i}, x'_{d,i} >\}, i = \{0, 1, \dots, N_g - 1\}$, initial set $\Omega_l = \emptyset$, the maximum number of iterations for each seed max_iter_l , the perturbation size of each iteration $step_size_l$, the decay rate of momentum μ_l , step size for random disturbance r_step_l .

Output: A set of IDI pairs found locally Ω_l .

```

1  For  $i = 0 : N_g - 1$ 
2    Select seed  $< x, x' >$  from  $\Omega_g$ ,  $g_0 = g'_0 = 0$ .
3    For  $t = 0 : max\_iter_l$ 
4      If ( $mod(step\_size_l, r\_step_l) == 0$ ) Then
5         $r = Rand_{(0,1)}(p_r)$ 
6      End If
7       $g_{t+1} = \mu_l \times g_t + \nabla_x J_{dl}(x; \Theta)$ 
8       $g'_{t+1} = \mu_l \times g'_t + \nabla_{x'} J_{dl}(x'; \Theta)$ 
9       $dire = sign(g_{t+1} + g'_{t+1})$ 
10      $p_{dire} = SoftMax(|g_{t+1} + g'_{t+1}|^{-1})$ 
11     For  $a^{ns} \in A_{ns}$ 
12       Generate a random number  $p_{tmp} \in (0, 1]$ .
13       If ( $p_{tmp} < p_{dire}[a^{ns}]$ ) Then
14          $x[a^{ns}] = x[a^{ns}] + dire[a^{ns}] \times step\_size_l$ 
15       End If
16     End For
17      $x = Clip(x, \mathbb{I})$ 
18     Create  $< x, x' >$  s.t.  $x[A_s] \neq x'[A_s], x[A_{ns}] = x'[A_{ns}]$ .
19     If ( $f(x; \Theta) \neq f(x'; \Theta)$ ) Then
20        $\Omega_l = \Omega_l \cup < x, x' >$ 
21     End If
22   End For
23 End For
```

19.4.2.1 Global Generation

To increase the IDIs' diversity, we design a dynamic loss as follows:

$$J_{dl}(x; \Theta) = -\frac{1}{N} \sum_{i=0}^{N-1} \sum_{k=1}^{N^k} [(p_k|r_k) \times f^k(x'_i; \Theta) \times \log(f^k(x_i; \Theta))], \quad (19.3)$$

where x'_i comes from x_i after flipping its sensitive attribute, N^k is the number of neurons in the most biased layer, f^k is the activation output of the k -th neuron. p is the position of biased neurons, and r is the position of randomly selected neurons to increase the dynamics of $J_{dl}(x; \Theta)$. $r = Rand_{(0,1)}(p_r)$, where $Rand_{(0,1)}(p_r)$ returns a random vector with only “0” or “1”. r has the same size as p and satisfies $\sum r = INT(N^k \times p_r)$, where $INT(\cdot)$ returns an integer. Here, we set $p_r = 5\%$. “|” means

“or”, $p_k | r_k = 0$ if and only $p_k = 0$ and $r_k = 0$. The optimization object of IDI generation is $\arg \max J_{dl}(x; \Theta)$.

Algorithm 19.2 shows the details of global generation with momentum acceleration. We first adopt k-means clustering function KMeans(X, N_c) to process X into N_c clusters, and then get seeds from clusters in a round-robin fashion at line 3. Lines 5 through 7, update random vector r at equal intervals, and r at equal intervals. This is not only to enhance the dynamic effect, but also to avoid excessive interference with the generation task. According to Definition 19.1, we determine the IDIs from lines 8 to 12. Lines 13 and 14 employ a momentum acceleration operation that effectively utilizes past gradients to reduce ineffective searches. Note that line 16 keeps the sensitive attribute value at x . Finally, we clip the value of x to satisfy the input domain \mathbb{I} .

19.4.2.2 Local Generation

Increase the number of iterations per seed max_iter_l , and reduce the bias perturbation at each iteration, since the goal of local generation is to find as many IDIs as possible that are close to the seed, as shown in Algorithm 19.3. The major difference compared to the global phase is the loop from line 11 to line 16, where perturbations are added with less probability to larger gradients of non-sensitive attributes. We automatically get the probability of adding perturbation to each attribute in x at line 10.

19.4.3 Generalization Framework on Unstructured Data

Our goal is to solve the problem of generalizing our approach to unstructured data by modifying A_s . Let’s take image data as an example. Attributes of an image are determined by pixels with normalized values between 0 and 1, i.e., the input domain of images is $\mathbb{I} \in [0, 1]$. Motivated by the adversarial attack, we design a generalization framework to implement the image’s A_s modification, which modifies A_s through adding a small perturbation to most pixels, as shown in Fig. 19.4.

To consider a face detection fairness test scenario which is used to determine whether the input image contains a face or not. The face detector consists of a CNN module (i.e., Fig. 19.4 (i)) and a FCN module (i.e., Fig. 19.4 (ii)). As shown in Fig. 19.4, for a given face image x and a detector $f(x; \Theta)$, there are three steps: ① build a sensitive attribute classifier; ② produce Δ_{senatt} based on Eq. (19.4); Δ_{senatt} is the perturbation added to image to flip sensitive attribute; ③ generate Δ_{bias} based on our method, where Δ_{bias} is the bias perturbation added to an image to flip the detection result.

First, we need a sensitive attribute classifier $f_{sa}(x; \Theta_{sa})$ that can distinguish the face image’s A_s (e.g., gender). We build the A_s classifier by adding a new FCN module (i.e., Fig. 19.4 (iii)) to the face detector’s CNN module (i.e., Fig. 19.4 (i)).

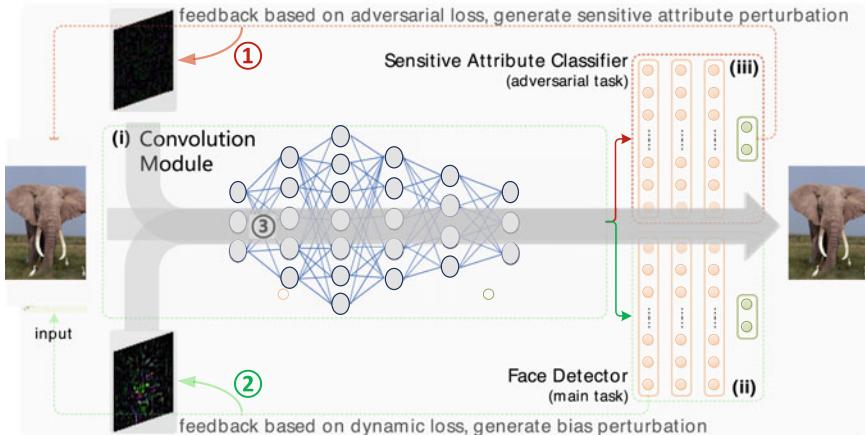


Fig. 19.4 An overview of generalization framework on image data type

Then, we froze the weights of the CNN module, and train the weights of the newly added FCN module.

Next, we modify the face image's A_s based on the adversarial attack. A classic adversarial attack FGSM [22] is adopted to flip the predicted result of the sensitive attribute by generating Δ_{senatt} as follows:

$$\Delta_{senatt} = \epsilon \times \text{sign}(\nabla_x J(x, y_{sa}; \Theta_{sa})) \quad (19.4)$$

satisfying that $f_{sa}(x; \Theta_{sa}) \neq f_{sa}(x + \Delta_{senatt}; \Theta_{sa})$,

where ϵ is a hyperparameter to determine perturbation size; $\text{sign}(\cdot)$ is a signum function return “-1”, “0”, or “1”; x is an input image; y_{sa} is the ground truth of x about sensitive attributes; and Θ_{sa} is the weights of the A_s classifier.

Finally, we leverage our method to generate Δ_{bias} , and then determine whether the instance pair $\langle x + \Delta_{bias}, x + \Delta_{bias} + \Delta_{senatt} \rangle$ satisfy Definition 19.1. First, the identification of each detection layer is determined. In the CNN module, the activation outputs of the convolutional layers are flattened. Then, during image IDI generation, only global generation is used due to the different data formats of images and structured data. Taking the input image in Fig. 19.4 as an example, its attributes can be regarded as $A \in \mathbb{R}^{64 \times 64 \times 3}$. In the local phase, many IDIs are developed from the seed IDIs images generated in the global phase. However, since the IDIs of these images are similar and differ by only a few pixels, they have little impact on improving the fairness of the face detector. Besides, we cancel the signum function $\text{sign}(\cdot)$ at line 15 of Algorithm 19.2 for image data.

For other unstructured data (e.g., text), we could first be processed into image-like vector structures using standard embedding techniques (e.g., word embedding), then apply the framework in Fig. 19.4 in a similar way. To ensure the generated inputs are realistic, we follow previous works (e.g., ADF [20] and EIDIG [21]) to limit the range

of perturbed features for structured data (e.g., maximum age limit) and the maximum perturbation size for unstructured data (e.g., human-imperceptible perturbation). The results of the distance evaluation show that the distance between the generated inputs and the seed inputs is relatively small, which means that the generated inputs are not surprising compared to the training data.

19.5 Experimental Setting

19.5.1 Datasets

We evaluate our method on seven datasets of which five are structured datasets and two are image datasets. Each dataset is divided into three parts, i.e., 70%, 10%, and 20% as training, validation, and testing, respectively.

The five open-source structured datasets include Adult, German credit (GerCre), bank marketing (BanMar), COMPAS, and medical expenditure panel survey (MEPS). Among them, Adult is applied to census income scenario, which contains three sensitive attributes of gender, race, and age, and has 48,842 records with a dimension of 13. GerCre and BanMar are both applied to credit scenario, and they both have age as a sensitive attribute, but GerCre has one more sensitive attribute of gender. GerCre has 1000 records with a dimension of 20. BanMar has 41,188 records with a dimension of 16. COMPAS is applied to the law scenario and contains the race-sensitive attribute with 5,278 records with a dimension of 400. MEPS is applied to the medical care scenario and contains the gender-sensitive attribute with 15,675 records with a dimension of 137. All datasets can be downloaded from GitHub¹ and preprocessed by AI Fairness 360 toolkit (AIF360) [34].

The two image datasets (i.e., ClbA-IN and LFW-IN) are constructed by us for face detection. They both have gender- and race-sensitive attributes, and their dimensions are $64 \times 64 \times 3$. ClbA-IN dataset consists of 60,000 face images from CelebA [35] and 60,000 non-face images from ImageNet [36]. LFW-IN dataset consists of 10,000 face images from LFW [37] and 10,000 non-face images from ImageNet [36]. The pixel value of each image is normalized to $[0,1]$.

19.5.2 Classifiers

We implement five FCN-based classifiers [20, 34] for structured datasets and two CNN-based face detectors [11, 38, 39] for image datasets since FCN and CNN are the most widely used basic structures in real-world classification tasks.

The five FCN-based classifiers can be divided into two types. The one is composed of five hidden layers for processing low-dimensional data (i.e., Adult, GerCre,

¹ <https://github.com/Trusted-AI/AIF360/tree/master/aif360/data>.

BanMar), denoted as LFCN. The another is composed of eight hidden layers for processing high-dimensional data (i.e., COMPAS and MEPS), denoted as HFCN. The activation functions in hidden layers and the output layer are ReLU and SoftMax, respectively. The hidden layer structures of LFCN and HFCN are [64, 32, 16, 8, 4] and [256, 256, 64, 64, 32, 32, 16, 8], respectively.

The two CNN-based face detectors serve for face detection, which are variants from two pre-trained models (i.e., VGG16 [38] and ResNet50 [39]) of *keras*. *applications*. We use the CNN module of VGG16 and ResNet50 as shown in Fig. 19.4 (i), and design the FCN module of Fig. 19.4 (ii) and (iii) as [512, 256, 128, 64, 16].

19.5.3 Baselines

We implement and compare four state-of-the-art (SOTA) methods with our method to evaluate their performance, including Aequitas [19], SymbGen [18], ADF [20], and EIDIG [21]. Note that Themis [17] has been shown to be significantly less effective for DNN and thus is omitted [18, 20]. We obtained the implementation of these baselines from GitHub.^{2,3} All baselines are set according to the optimal performance settings indicated in the relevant papers.

19.5.4 Evaluation Metrics

Five aspects of our method are evaluated, including generation *effectiveness*, *efficiency*, *interpretability*, the *utility* of AUC metric, and *generalization* of our method.

19.5.4.1 Generation Effectiveness Evaluation

We evaluate the effectiveness of our method on structured data from two aspects: generation quantity and quality.

(1) **Quantity.** To evaluate the generation quantity, we first count the total number of IDIs, then count the global IDIs' number and local IDIs' number, respectively, recorded as “#IDIs”. Note that the duplicate instances are filtered.

(2) **Quality.** We use generation success rate (GSR), generation diversity (GD), and IDIs' contributions to fairness improvement (i.e., DM-RS [19–21]) to evaluate IDIs' quality.

$$\text{GSR} = \frac{\# \text{ IDIs}}{\# \text{ non-duplicate instances}} \times 100\%, \quad (19.5)$$

² <https://github.com/pxzhang94/ADF>.

³ <https://github.com/LingfengZhang98/EIDIG>.

where non-duplicate instances represent the input space.

$$\text{GD}_{NF}(\rho_{\text{cons}}, \text{baseline}) = \frac{CR_{NF-bl}}{CR_{bl-NF}}, \quad (19.6)$$

where $CR_{NF-bl} = \frac{\# \text{ IDIs of baselines fall in } \Pi_{NF}}{\# \text{ IDIs of baseline}}$ represents the coverage rate of our method's IDIs to baseline's IDIs, Π_{NF} is the area with our method's IDIs as the center and cosine distance ρ_{cons} as the radius; similar to $CR_{bl-NF} = \frac{\# \text{ IDIs of ourmethod fall in } \Pi_{bl}}{\# \text{ IDIs of ourmethod}}$. Our method's IDIs are more diverse when $\text{GD}_{NF} > 1$.

By retraining the DNN with these IDIs, the generated IDIs improve the fairness of the DNN. DM-RS is the percentage of IDIs in randomly sampled instances. The higher the DM-RS value, the more biased the DNN is, i.e., the less the IDI contributes to improving fairness.

$$\text{DM-RS} = \frac{\# \text{ IDIs}}{\# \text{ instances randomly sampled}} \times 100\%. \quad (19.7)$$

19.5.4.2 Interpretability Evaluation Based on Biased Neurons

To interpret the utility of our method, we refer to paper [40] to design the coverage of biased neurons, which is defined as follows: for a given instance, compute the activation output of the most biased layer; normalize the activation values; select neurons with activation values greater than 0.5 as the activated neurons; and compare the coverage of the activated neurons to the biased neurons.

19.5.4.3 Utility Evaluation of AUC Metric

In our work, AUC values are used as a measure of discriminatory power based on interpreted results. We evaluate the utility of AUC metrics from three aspects: consistency, significance, and complexity between AUC and DM-RS.

(1) **Consistency.** To evaluate the consistency, we adopt Spearman's correlation coefficient, as follows:

$$\rho_s = 1 - \frac{6 \sum_{i=1}^n d_i^2}{n(n^2 - 1)}, \quad (19.8)$$

where $d_i = a_i - b_i$, $i \in \{1, 2, \dots, n\}$, a_i and b_i are the rank of AUC and DM-RS values, respectively. High ρ_s means more consistent.

(2) **Significance.** To evaluate whether AUC can measure discrimination more significantly than DM-RS, we use the standard deviation, as follows:

$$\sigma = \sqrt{\frac{\sum_{i=1}^n n(c_i - \mu)^2}{n - 1}}, \quad (19.9)$$

where c_i is AUC or DM-RS of different testing methods, $i \in \{1, 2, \dots, n\}$, μ is the mean value of c_i . Large σ means more significant.

19.5.4.4 Generalization Evaluation on Image Data

The generalization of our method with respect to image data is evaluated from two aspects: generation quantity, and quality.

(1) **Quantity.** To evaluate the generation quantity on image data, only the overall number of IDIs in the image recorded as “#IDIs” is counted.

(2) **Quality.** To evaluate the quality of IDI, we used the contribution of GSR and IDI to the unbiased improvement of the face detector based on AUC values to calculate the detection rate (DR) after retraining.

19.6 Experimental Results

We evaluate our method through answering the following two research questions (RQ): (1) how *effective* is our method; (2) how to *interpret* the utility of our method;

19.6.1 Research Questions 1

How effective is our method in generating IDIs?

When reporting the results, we focus on the following aspects: generation *quantity* and *quality*.

Generation Quantity. The evaluation results are shown in Figs. 19.5, 19.6, and 19.7, including three scenarios: the *total* number of IDIs, the IDIs number in *global* phase, and the IDIs number in *local* phase.

Implementation details for quantity evaluation: (1) SymbGen works differently from other baselines, thus we follow the comparison strategy of Zhang et al. [20], i.e., evaluating the generation quantity of our method and SymbGen within the same time (i.e., 500 sec) limit, as shown in Fig. 19.6; (2) for a fair global phase comparison, we generate 1,000 non-duplicate instances without constrained by num_g , then count IDIs number and record it in Fig. 19.7, where the seeds used are consistent for different methods; (3) for a fair local phase comparison, we mix IDIs generated globally by different methods, and randomly sample 100 as the seeds in local phase; then generate 1,000 non-duplicate instances for each seed without constrained by max_iter_l , count the IDIs number on average for each seed, and record it in Fig. 19.7.

- Our method generates more IDIs than baselines, especially for densely coded structured data. For instance, in Fig. 19.5, on Adult dataset with different attributes, the IDIs’ number of our method is 217,855 on average, which is 16.5 times and 1.6 times that of Aequitas and EIDIG, respectively. In addition, in Fig. 19.6, our method generates much more IDIs than SymbGen on all datasets. The outstanding performance of our method is mainly because the optimization object of our method takes into account the whole DNNs’ discrimination information through

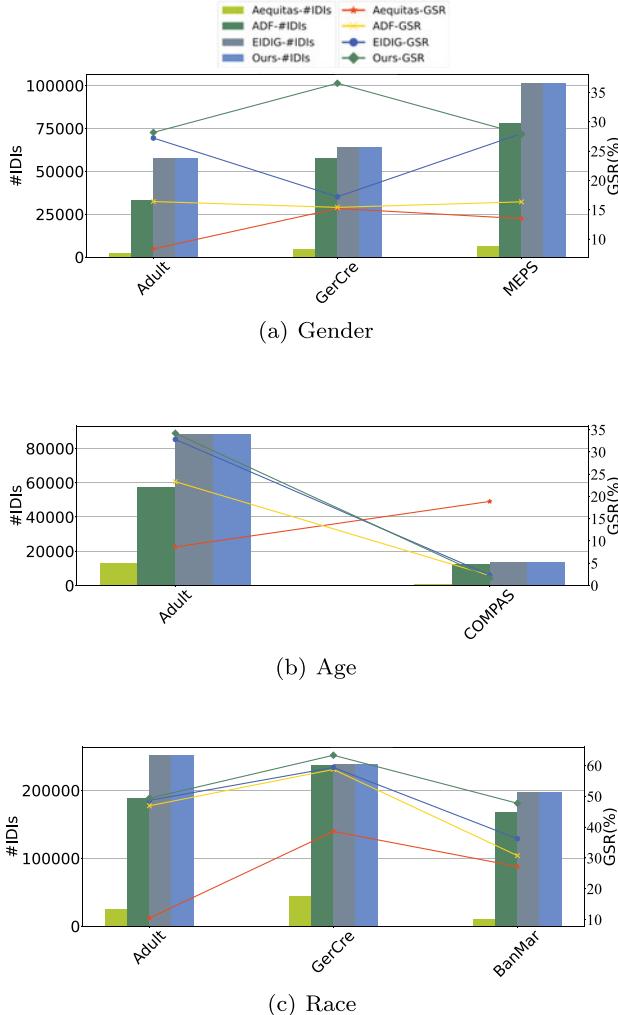


Fig. 19.5 Comparison with Aequitas, ADF, and EIDIG based on the total number of generated IDIs and GSR.

the biased neurons while Aequitas and EIDIG only depend on the output layer. However, the IDIs' number on COMPAS dataset with race attribute is 11,232, which is slightly lower than that of EIDIG. Since the COMPAS is encoded as one-hot in AIF360 [34], we speculate the reason is that too sparse data coding reduces the derivation efficiency from biased neurons.

- As shown in Fig. 19.7, our method generates much more IDIs than all baselines in the global phase, which is beneficial to increase the diversity of our method's

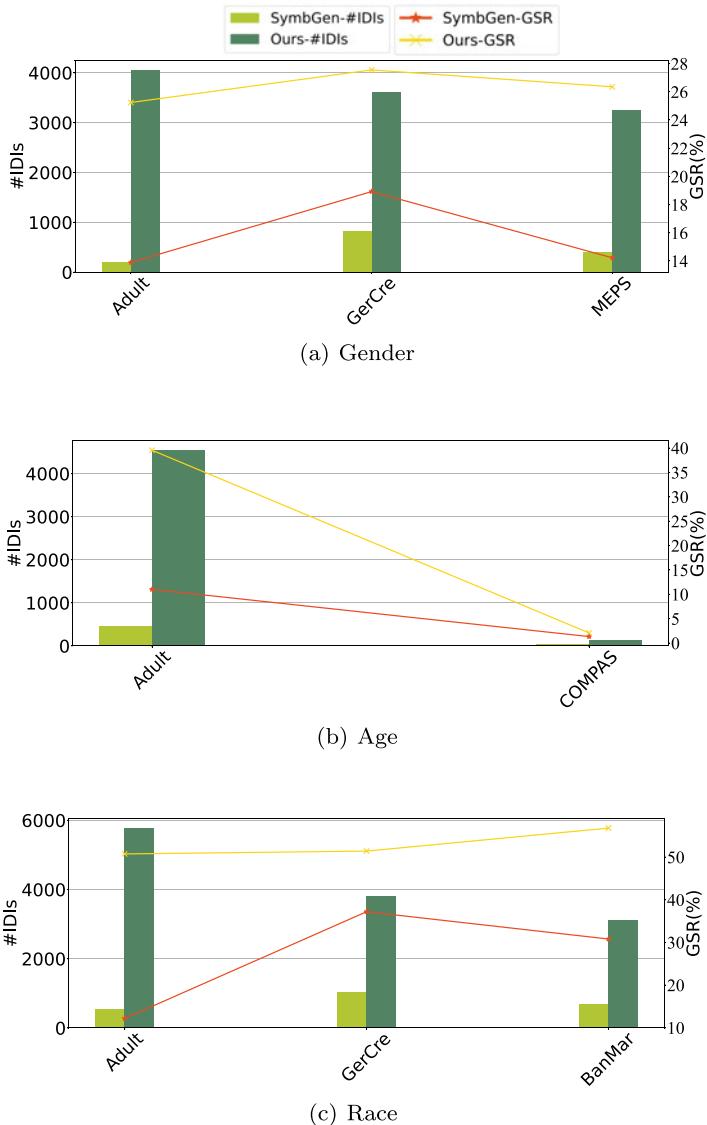


Fig. 19.6 Comparison with SymbGen based on the number of IDIs generated in 500 s

IDIs in the subsequent local phase. For instance, on all datasets, the IDIs' number of our method is 866 on average, which is 9.45 times and 1.42 times that of Aequitas and EIDIG, respectively. This is mainly because the optimization object of our method takes into account the dynamics through the dynamic combination

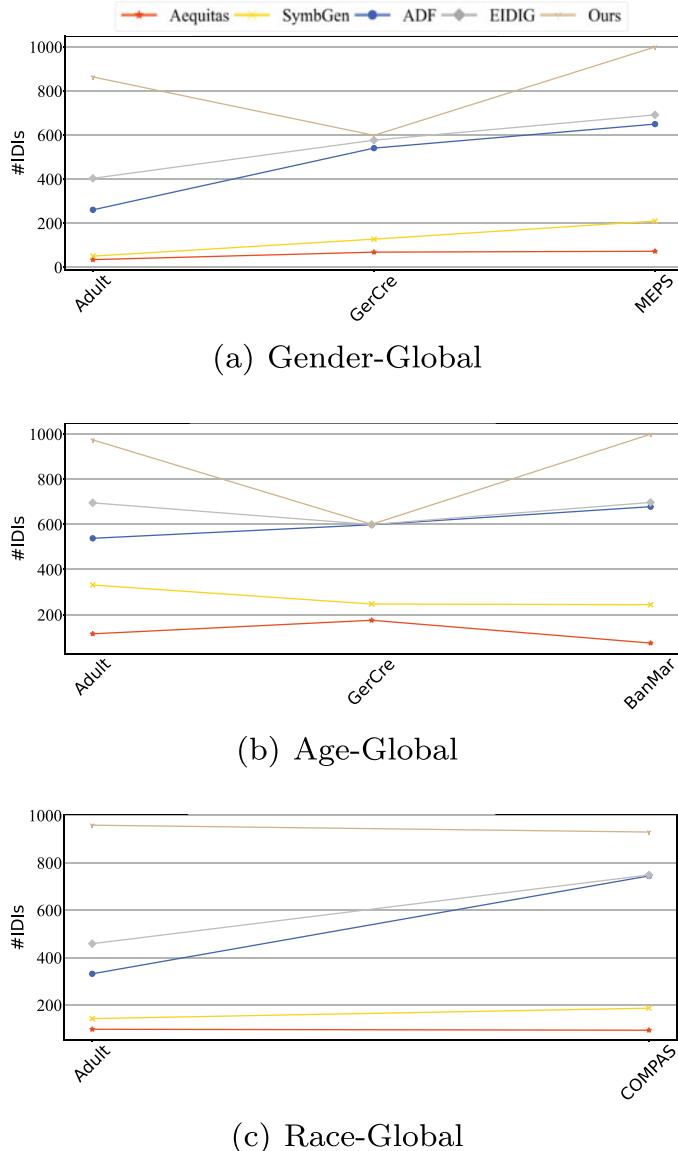
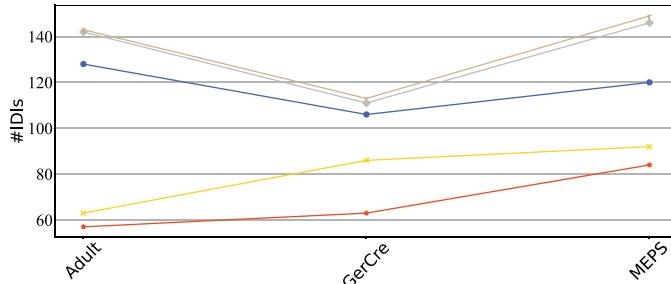
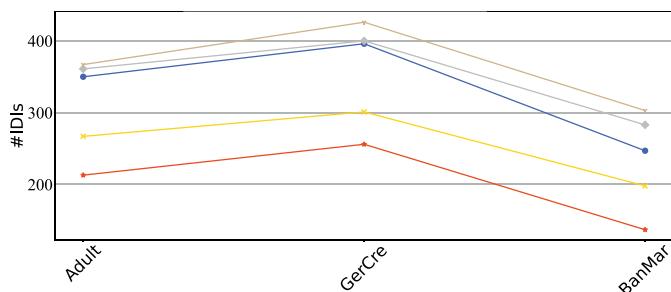


Fig. 19.7 “#IDIs” measurement in the global and local phases

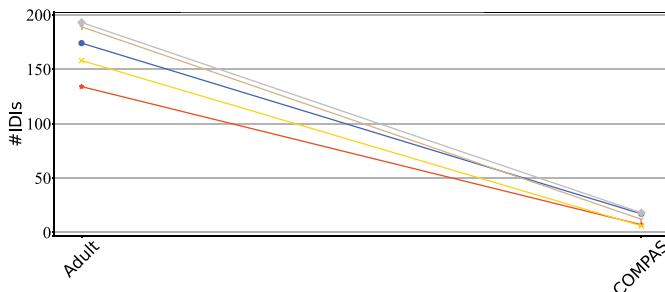
of biased neurons. Thus, our method searches a larger space to generate more global IDIs. Besides, we conduct a preliminary T-test about “#IDIs” on the Adult and GerCre datasets. The p-values of all models are small enough to reject the null hypothesis (i.e., less than 0.05), which demonstrates the significance of our method.



(d) Gender-Local



(e) Age-Local



(f) Race-Local

Fig. 19.7 (continued)

- In the local phase, our method is much more efficient than baselines in general. For instance, in Fig. 19.7, on average, our method returns 78.97%, 45.35%, 10.81%, and 2.90% more IDIs than Aequitas, SymbGen, ADF, and EIDIG, respectively. Recall that Aequitas, ADF, EIDIG, and our method all guide local phase through a probability distribution, which is the likelihood of IDIs by modifying several certain attributes (i.e., the loop from lines 11 to 16 of Algorithm 19.3). The proba-

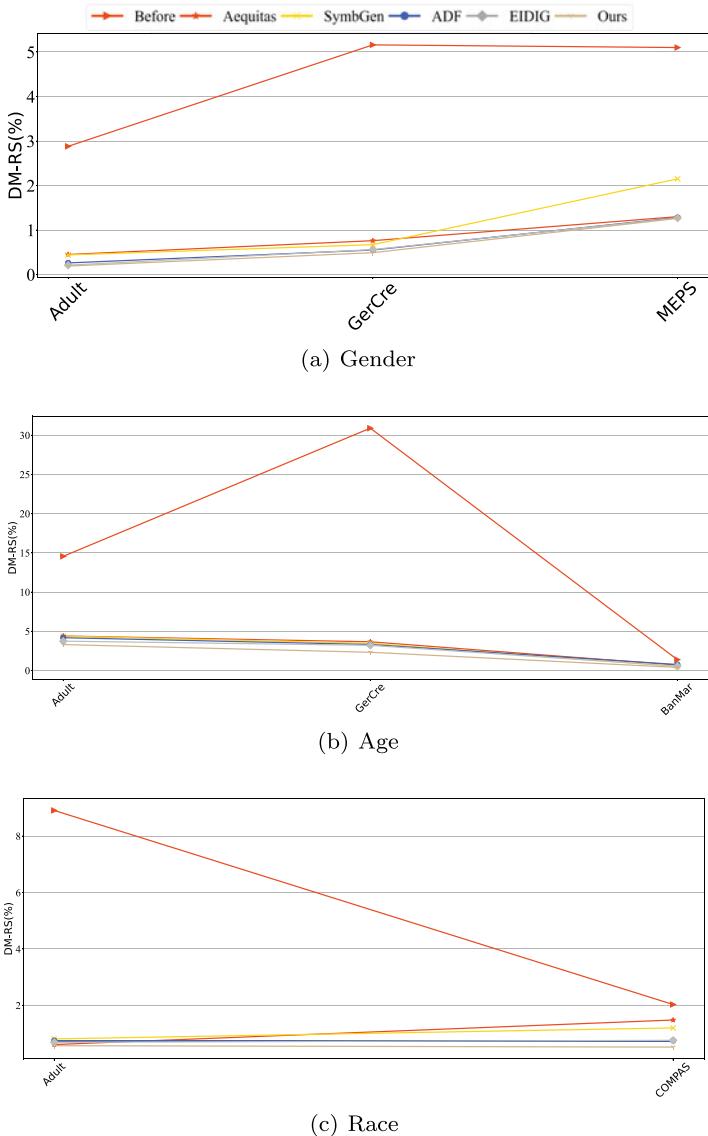


Fig. 19.8 Fairness improvement measured by DM-RS, where “Before” and “After” represent the original and the retrained DNNs, respectively

bility determination of our method takes into account the momentum and SoftMax activation (i.e., at line 10 of Algorithm 19.3) while the baselines do not. Hence, our method generates more local IDIs.

Generation Quality. The evaluation results are shown in Figs. 19.5, 19.6, 19.8, Tables 19.1, and 19.2, including the generation success rate (*GSR*), generation diversity (*GD*), and fairness improvement (*DM-RS*).

Table 19.1 Generation diversity of our method compared to Aequitas

$\rho_{\{cons\}}$	$GD_{\{NF\}}(\rho_{\{cons\}}, Aequitas)$			
	L1(Adult, gender)	L2(Adult, race)	L3(GerCre, gender)	L4(GerCre, age)
0.01	6.66	9.79	4.43	5.56
0.1	2.11	3.33	1.51	1.08
0.2	1.10	2.20	1.42	0.32
0.3	0.02	1.10	1.33	0.09
0.4	0.00	0.00	0.00	0.00

Table 19.2 Generation diversity of our method compared to ADF

$\rho_{\{cons\}}$	$GD_{\{NF\}}(\rho_{\{cons\}}, ADF)$			
	L1(Adult, gender)	L2(Adult, race)	L3(GerCre, gender)	L4(GerCre, age)
0.005	2.12	2.23	1.30	1.04
0.02	1.26	1.38	1.10	1.00
0.04	1.10	1.01	1.19	1.00
0.06	1.08	1.01	1.13	1.00
0.08	1.07	1.01	1.13	1.00
0.1	1.06	1.00	1.11	1.00

Implementation details for quality evaluation: (1) for a fair diversity comparison, we seed each method with the same set of 10 global IDIs and apply them to generate 100 local IDIs for each seed without considering \max_iter_l , as shown in Tables 19.1, and 19.2; (2) we randomly select 10% IDIs of each method to retrain DNNs, then compute their fairness improvement results; to avoid contingency, we repeat five times and record the average DM-RS value in Fig. 19.8.

- As shown in Figs. 19.5 and 19.6, the GSR values of our method are higher than that of baselines on almost all datasets, i.e., our method can search for a larger valid input space, where the input space is calculated by “#IDIs/GSR”. For instance, in Fig. 19.5, on all datasets, Aequitas has a GSR of 17.62% on average, whereas our method achieves a GSR of 36.12%, which is $\sim \times 2.1$ more than that of Aequitas. The outstanding performance of our method is mainly because it not only considers the whole DNN’s discrimination through biased neurons, but also takes into account the dynamics of the optimization object through the combination of biased neurons. Thus, our method searches a larger valid input space than Aequitas.

Meanwhile, the GSR value of our method on different sensitive attributes is more robust than ADF and EIDIG. For instance, in Fig. 19.5, on the GerCre dataset with gender and age attributes, the GSR values of our method are 36.57% and 63.35%, whereas that of EIDIG are 17.23% and 59.38%. We hypothesize that the reason for this is that the identification of gender attributes is not evident in the output layer. Our method can identify potential fairness violations by analyzing the internal discrimination of biased neurons, allowing for stable testing across different sensitive attributes.

In addition, the valid input space of our method is larger than baselines in general, i.e., a larger input space supports more diverse IDI generation. For instance, in Fig. 19.6, the average input space of our method is 3.30 times that of SymbGen. This is mainly due to the fact that the momentum acceleration strategy utilizes the past gradient as a secondary guide, reducing the number of invalid searches. Hence, our method generates more IDIs in a large input space.

- In all cases, our method generates more diverse IDIs, which facilitates the identification of more potential discrimination and improves equity through re-education. For instance, in Table 19.1 and Table 19.2, compare to Aequitas and ADF, the GD_{NF} values are all greater than “1” under different radius values ρ_{cons} , and as the radius increases, the value of GD_{NF} gradually converges to “1”. The results show that the IDIs generated by our method always cover the baseline IDIs. The reason for this is probably that the dynamic loss function extends the effective input space by combining neurons with different biases as optimization targets.

Besides, a close investigation shows that there is a similar trend in the generation diversity for the same sensitive attribute. For instance, when ρ_{cons} (0.1 in Table 19.1 or ρ_{cons} (0.02 in Table 19.2, the “L2” with race is always the highest, the “L4” with age is always the lowest, while “L1” and “L3” with gender are in the middle. Since both datasets Adult and GerCre are linked to financial aspects such as income and debt, they share a common theme, we speculate that there is similar discrimination for gender in classifiers LFC-A and LFC-G for similar tasks, thus GD_{NF} shows similar trends in gender attribute.

- Our method is capable of obtaining smaller DM-RS values in all cases, i.e., our method generates IDIs that make a greater contribution towards improving the fairness of DNNs. For instance, in Fig. 19.8, measured by DM-RS, our method has shown an average improvement of 87.24% in fairness compared to the baselines, which are 81.18% for Aequitas, 80.78% for SymbGen, 83.30% for ADF, and 84.49% for EIDIG. This is because our method’s IDIs (individualized differential impact) are more diverse than the baselines, which allows us to discover more potential fairness violations and implement higher fairness improvement through retraining.

Answer to RQ1: Our method outperforms the SOTA methods (i.e., Aequitas, SymbGen, ADF, and EIDIG) in two aspects: (1) *quantity*—it generates $\sim \times 5.84$ IDIs on average compared to baselines; (2) *quality*—it searches $\sim \times 3.03$ input space with more than $\sim \times 1.65$ GSR on average compared to baselines, it generates IDIs that are $\sim \times 6.24$ and $\sim \times 1.38$ more diverse than Aequitas and ADF on average with ρ_{cons} (0.02, it is beneficial to DNNs’ fairness improvement of 87.24% on average.

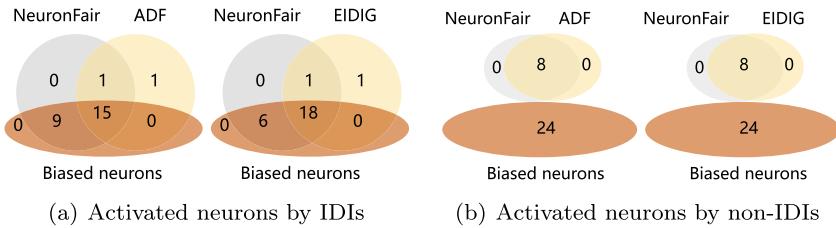


Fig. 19.9 The overlap of biased neurons and neurons activated by different instances

19.6.2 Research Questions 2

How to interpret our method's utility by biased neurons?

To understand the utility, we refer to the biased neuron *coverage*. The evaluation results are shown in Fig. 19.9.

Here are the implementation details for interpretation of our method: (1) we conducted experiments on the Adult dataset, which includes a gender attribute, using the LFC-A classifier; (2) we compare the interpretation results of our method with ADF and EIDIG; (3) for a fair interpretation, we randomly select 10% IDIs and 10% non-IDIs (i.e., the generated failure instances) for each method, and then compute the coverage of biased neurons, as shown in Fig. 19.9.

- Our method can make use of biased neurons to interpret the usefulness of IDIs. First, the activation of biased neurons by IDIs can lead to discrimination. For instance, the neurons activated by IDIs can cover most of the biased neurons in Fig. 19.9a, while the coverage of the biased neurons by non-IDIs of different methods is 0 in Fig. 19.9b. We can further interpret the utility of testing methods is related to the coverage of biased neurons, i.e., our method outperforms ADF and EIDIG because it identifies discrimination in all neurons. For instance, in Fig. 19.9a, our method's IDIs activate all 24 biased neurons in the 2-nd layer of LFC-A classifier, while the neurons activated by other IDIs cannot cover all (15 for ADF and 18 for EIDIG).

Answer to RQ2: The main reason for our method's utility is that its IDIs can activate more biased neurons. Our method's IDIs activate 100% biased neurons, while 62.5% for ADF and 75% for EIDIG.

19.7 Conclusions

We have developed a method called NeuronFair that tests the fairness of deep neural networks (DNNs) in an interpretable and efficient way. Our method offers a clear interpretation of discrimination and generates IDI for various data formats. In the discrimination interpretation, AS curve and AUC measurement are used to interpret

discrimination severity in each DNN layer. In the IDI generation, a global phase and a local phase work together to systematically search the input space for IDIs, guided by momentum acceleration and dynamic loss. Further, our method can process not only structured data but also unstructured data, e.g., image, text, etc. We compare our method with four SOTA methods in five structured datasets and two face image datasets against seven DNNs, the results show that our method has significantly better performance in terms of interpretability, generation effectiveness, and data generalization.

References

1. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
2. Meng, L., Li, Y., Chen, L., Wang, Z., Wu, D., Zhou, Y., Xu, B.: Measuring discrimination to boost comparative testing for multiple deep learning models. In: 43rd IEEE/ACM International Conference on Software Engineering. ICSE 2021, Madrid, Spain, 22–30 May 2021, pp. 385–396. Piscataway, IEEE (2021)
3. Ramadan, Q., Ahmadian, A.S., Strüber, D., Jürjens, J., Staab, S.: Model-based discrimination analysis: a position paper. In: Brun, Y., Johnson, B., Meliou, A. (eds.) Proceedings of the International Workshop on Software Fairness, FairWare@ICSE 2018, Gothenburg, Sweden, May 29, 2018, pp. 22–28. ACM, New York (2018)
4. Zhang, J.M., Harman, M.: “Ignorance and prejudice” in software fairness. In: 43rd IEEE/ACM International Conference on Software Engineering. ICSE 2021, Madrid, Spain, 22–30 May 2021, pp. 1436–1447. Piscataway, IEEE (2021)
5. Aggarwal, A., Lohia, P., Nagar, S., Dey, K., Saha, D.: Black box fairness testing of machine learning models. In: Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26–30, 2019, pp. 625–635. ACM, New York (2019)
6. Kohavi, R.: Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, pp. 202–207. AAAI Press, Menlo Park (1996)
7. Huang, C., Zhang, J., Zheng, Y., Chawla, N.V.: Deepcrime: attentive hierarchical recurrent networks for crime prediction. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22–26, 2018, pp. 1423–1432. ACM, New York (2018)
8. Gaur, B., Saluja, G.S., Sivakumar, H.B., Singh, S.: Semi-supervised deep learning based named entity recognition model to parse education section of resumes. *Neural Comput. Appl.* **33**, 5705–5718 (2021)
9. Mai, F., Tian, S., Lee, C., Ma, L.: Deep learning models for bankruptcy prediction using textual disclosures. *Eur. J. Oper. Res.* **274**(2), 743–758 (2019)
10. Buolamwini, J., Gebru, T.: Gender shades: intersectional accuracy disparities in commercial gender classification. In: Conference on Fairness, Accountability and Transparency, FAT 2018, 23–24 February 2018, New York, NY, USA. Proceedings of Machine Learning Research, vol. 81, pp. 77–91. PMLR, Stockholm, Sweden (2018)
11. Ramaswamy, V.V., Kim, S.S.Y., Russakovsky, O.: Fair attribute classification through latent space de-biasing. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19–25, 2021, pp. 9301–9310. Computer Vision Foundation/IEEE (2021)
12. Farahani, A., Pasquale, L., Bennaceur, A., Welsh, T., Nuseibeh, B.: On adaptive fairness in software systems. In: 16th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. SEAMS@ICSE 2021, Madrid, Spain, May 18–24, 2021, pp. 97–103. Piscataway, IEEE (2021)

13. Clegg, B.S., North, S., McMinn, P., Fraser, G.: Simulating student mistakes to evaluate the fairness of automated grading. In: Beecham, S., Damian, D.E. (eds.) Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training, ICSE (SEET) 2019, Montreal, QC, Canada, May 25–31, 2019, pp. 121–125. IEEE/ACM, Piscataway (2019)
14. Germán, D.M., Robles, G., Poo-Caamaño, G., Yang, X., Iida, H., Inoue, K.: “Was my contribution fairly reviewed?”: a framework to study the perception of fairness in modern code reviews. In: Chaudron, M., Crnkovic, I., Chechik, M., Harman, M. (eds.) Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27–June 03, 2018, pp. 523–534. ACM, New York (2018)
15. Dwork, C., Hardt, M., Pitassi, T., Reingold, O., Zemel, R.S.: Fairness through awareness. In: Innovations in Theoretical Computer Science 2012. Cambridge, MA, USA, January 8–10, 2012, pp. 214–226. ACM, New York (2012)
16. Black, E., Yeom, S., Fredriksson, M.: Fliptest: fairness testing via optimal transport. In: FAT* ’20: Conference on Fairness, Accountability, and Transparency, Barcelona, Spain, January 27–30, 2020, pp. 111–121. ACM, New York (2020)
17. Galhotra, S., Brun, Y., Meliou, A.: Fairness testing: testing software for discrimination. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4–8, 2017, pp. 498–510. ACM, New York (2017)
18. Aggarwal, A., Lohia, P., Nagar, S., Dey, K., Saha, D.: Automated test generation to detect individual discrimination in AI models. CoRR **abs/1809.03260**, 1–8 (2018)
19. Udeshi, S., Arora, P., Chattopadhyay, S.: Automated directed fairness testing. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3–7, 2018, pp. 98–108. ACM, New York (2018)
20. Zhang, P., Wang, J., Sun, J., Dong, G., Wang, X., Wang, X., Dong, J.S., Dai, T.: White-box fairness testing through adversarial sampling. In: ICSE ’20: 42nd International Conference on Software Engineering. Seoul, South Korea, 27 June–19 July, 2020, pp. 949–960. ACM, New York (2020)
21. Zhang, L., Zhang, Y., Zhang, M.: Efficient white-box fairness testing through gradient search. In: Cadar, C., Zhang, X. (eds.) ISSTA ’21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11–17, 2021, pp. 103–114. ACM, New York (2021)
22. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–10. Arxiv, [C/OL, 2015-5-20] (2015)
23. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Workshop Track Proceedings, pp. 1–14. OpenReview.net, [C/OL, 2017-2-11] (2017)
24. Chen, J., Zheng, H., Xiong, H., Shen, S., Su, M.: Mag-gan: Massive attack generator via gan. Inf. Sci. **536**, 67–90 (2020)
25. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30–May 3, 2018, pp. 1–12. OpenReview.net, [C/OL, 2018-2-16] (2018)
26. Chen, P., Zhang, H., Sharma, Y., Yi, J., Hsieh, C.: Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017, pp. 15–26. ACM, New York (2017)
27. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18–22, 2018, pp. 9185–9193. IEEE Computer Society, Computer Society Los Alamitos (2018). <http://dx.doi.org/10.1109/CVPR.2018.00957>

28. Tabacof, P., Valle, E.: Exploring the space of adversarial images. In: 2016 International Joint Conference on Neural Networks, IJCNN 2016, Vancouver, BC, Canada, July 24–29, 2016. vol. 2016-October, pp. 426–433. IEEE, Piscataway (2016)
29. Tramèr, F., Atlidakis, V., Geambasu, R., Hsu, D.J., Hubaux, J., Humbert, M., Juels, A., Lin, H.: Fairtest: Discovering unwarranted associations in data-driven applications. In: 2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26–28, 2017, pp. 401–416. Piscataway, IEEE (2017)
30. Adebayo, J., Kagal, L.: Iterative orthogonal feature projection for diagnosing bias in black-box models. CoRR **abs/1611.04967**, 1–5 (2016)
31. Kim, B., Wattenberg, M., Gilmer, J., Cai, C.J., Wexler, J., Viégas, F.B., Sayres, R.: Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10–15, 2018. Proceedings of Machine Learning Research, vol. 80, pp. 2673–2682. PMLR, Stockholm, Sweden (2018)
32. Du, M., Yang, F., Zou, N., Hu, X.: Fairness in deep learning: a computational perspective. CoRR **abs/1908.08843**, 1–9 (2019)
33. Liu, Y., Lee, W., Tao, G., Ma, S., Aafer, Y., Zhang, X.: ABS: scanning neural networks for backdoors by artificial brain stimulation. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11–15, 2019, pp. 1265–1282. ACM, New York (2019)
34. Bellamy, R.K.E., Dey, K., Hind, M., Hoffman, S.C., Houde, S., Kannan, K., Lohia, P., Martino, J., Mehta, S., Mojsilovic, A., Nagar, S., Ramamurthy, K.N., Richards, J., Saha, D., Sattigeri, P., Singh, M., Varshney, K.R., Zhang, Y.: AI Fairness 360: an extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias (2018)
35. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7–13, 2015, pp. 3730–3738. IEEE Computer Society, Computer Society Los Alamitos (2015)
36. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20–25 June 2009, Miami, Florida, USA. vol. 1–4, pp. 248–255. IEEE Computer Society, Computer Society Los Alamitos (2009)
37. Huang, G.B., Mattar, M., Berg, T., Learned-Miller, E.: Labeled faces in the wild: a database for studying face recognition in unconstrained environments. In: Workshop on Faces in “Real-Life” Images: Detection, Alignment, and Recognition, pp. 1–15. Erik Learned-Miller and Andras Ferencz and Frédéric Jurie, HAL-inria, Marseille, France (2008)
38. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7–9, 2015, pp. 1–14. Arxiv, [C/OL, 2015-4-10] (2015)
39. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27–30, 2016, pp. 770–778. IEEE Computer Society, Computer Society Los Alamitos (2016)
40. Pei, K., Cao, Y., Yang, J., Jana, S.: Deepxplore: automated whitebox testing of deep learning systems. Commun. ACM **62**(11), 137–145 (2019)

Chapter 20

A Deep Learning Framework for Dynamic Network Link Prediction



20.1 Introduction

Real-world systems are often modeled as dynamic networks, where nodes may come and go and links may vanish and recover over time. Links, representing the interactions between different entities, are crucial in the analysis of these networks. Networks are widely used to describe complex systems in various fields such as social science [1, 2], biology [3], electric system [4], and economics [5].

Link prediction in dynamic networks [6, 7] aims to predict the future structure of the network based on historical data. This helps us better understand network evolution and further investigate relationships between topologies and functions. For instance, in online social networks [8–10], we can predict which links are going to be established in the near future. It means that we can infer with what kind of people or even which particular one, the target user is likely to make friends based on their historical behaviors. Link prediction has also been applied to studies on disease contagions [11], protein–protein interactions [12], and many other fields where the evolution matters.

Although several works have been proposed to predict dynamic links in networks, most of them still lack the ability to adapt to dynamic networks. Temporal information is often ignored by models designed for static networks. Some recent works have attempted to address this issue by redefining common neighbors [13] and designing temporal walks for dynamic networks [14, 15]. With the development of deep learning [16, 17], many end-to-end frameworks have been proposed for dynamic network link prediction, which are not limited to homogeneous networks. For example, Ozcan et al. [18] and Ozcan et al. [19] have explored the link prediction problem on heterogeneous dynamic networks. However, these methods still face challenges such as dealing with changes in node and edge dynamics, incorporating additional contextual information, and handling spatiotemporal variations in the data.

The problem of predicting the global structure of networks in the near future, with a focus on links that are going to appear or disappear, has been addressed in this paper. Our method for link prediction in dynamic networks is proposed, which

takes advantage of the encoder–decoder architecture and a stacked Long Short-Term Memory (LSTM) module. The model can effectively handle high dimensionality, non-linearity, and sparsity. The encoder–decoder architecture allows the model to automatically learn representations of networks and reconstruct a graph based on the extracted information. The stacked LSTM module placed behind the encoder can learn relatively low-dimensional representations of sequences of graphs. To address the issue of network sparsity, existing links are amplified at the training process, encouraging the model to account for existing links more than missing or nonexistent ones. Comprehensive experiments were conducted on five real-world datasets, demonstrating that the proposed model significantly outperforms current state-of-the-art methods. The main contributions of this study are as follows:

- A general end-to-end deep learning framework for link prediction in dynamic networks has been proposed. The model automatically learns representations of networks and enhances the ability to learn temporal features through the encoder–decoder architecture and stacked LSTM module.
- Our method is competent to perform long-term prediction tasks with only a slight drop in performance. The model is suitable for networks of different scales by fine-tuning the model structure, such as changing the number of units in different layers. Additionally, the model can predict links that are going to appear or disappear, whereas most existing methods only focus on predicting the former.
- We introduce a new metric, called Error Rate, to comprehensively evaluate the performance of dynamic network link prediction. This metric is intended to complement the existing evaluation methods such as Area Under the ROC Curve (AUC), providing a more holistic view of the model’s accuracy.
- We have conducted extensive experiments by comparing our method with five baseline methods on various metrics. The results indicate that our model outperforms the others and achieves state-of-the-art performance in dynamic network link prediction.

20.2 Related Work

Random walk-based methods. Recent studies have demonstrated the applicability of random walks on dynamic networks. Ahmed et al. [20] assigned damping weights to each snapshots, prioritizing more recent ones, and combined them into a weighted graph to perform local random walk. This approach was further extended by Ahmed and Chen [14], who proposed Time-Series Random Walk (TS-RW) to integrate temporal and global information. In another study, Yu et al. [21] used random walk as a sampling method before passing the network into deep neural networks. Nguyen et al. [15] focused on continuous-time dynamic networks instead of discrete snapshots and proposed temporal walk to learn time-dependent embedding vectors without losing information.

Factorization-based methods. Incremental singular value decomposition (IncSVD) [22] is a method that uses a perturbation matrix to capture dynamics and modifies the SVD while theoretically guiding maximum-error-bounded restart of SVD (TIMERS) [23] which focuses on the adjacent matrix. Ma et al. [24] proposed a novel graph regularized non-negative matrix factorization algorithm (GrNMF) that factorizes the current graph with previous graphs used as regularization.

Deep learning-based methods. The evolution of a network can be considered a special case of a Markov random field with two-layer variables. Conditional temporal restricted Boltzmann machine (ctRBM) [25], also known as ctRBM, takes into account not only the neighboring connections but also the temporal connections. This allows it to predict future linkage status. In fact, the neighborhood information can be better utilized by incorporating temporal RBM and gradient boosting decision tree (GBDT) to model the evolution pattern of each link [26].

20.3 Methodology

20.3.1 Problem Definition

Definition 20.1 (*Dynamic Networks*) Given a sequence of graphs, $\{G_1, \dots, G_T\}$, where $G_k = (V, E_k)$ denotes the k th snapshot of a dynamic network. Let V be the set of all vertices and $E_k \subseteq V \times V$ the temporal links within the fixed timespan $[t_{k-1}, t_k]$. The adjacency matrix of G_k is denoted by A_k with the element $a_{k;i,j} = 1$ if there is a directed link from v_i to v_j and $a_{k;i,j} = 0$ otherwise.

In a static network, link prediction aims to identify edges that actually exist based on the observed distribution of edges. Similarly, in a dynamic network, link prediction makes use of information extracted from previous graphs to reveal the underlying evolving patterns of the network and predict its future status. Since the adjacency matrix can accurately describe the structure of a network, it is ideally suited to be used as input and output for the prediction model. The evolution of a network can be seen as a special case of a Markov random field with two-layer variables, where conditional temporal restricted Boltzmann machine (ctRBM), also known as ctRBM, takes into account not only the neighboring connections but also the temporal connections. This allows it to predict future linkage status by incorporating temporal RBM and gradient boosting decision tree (GBDT) to model the evolution pattern of each link. We could infer G_t just based on G_{t-1} , due to the strong relationship between the successive snapshots of the dynamic network. However, the information contained in G_t may be too little to do precise inference. In fact, not only the structure itself but also the structure change overtime matters in the network evolution. Thus, we prefer to use a sequence of length N , i.e., $\{G_{t-N}, \dots, G_{t-1}\}$, to predict G_t .

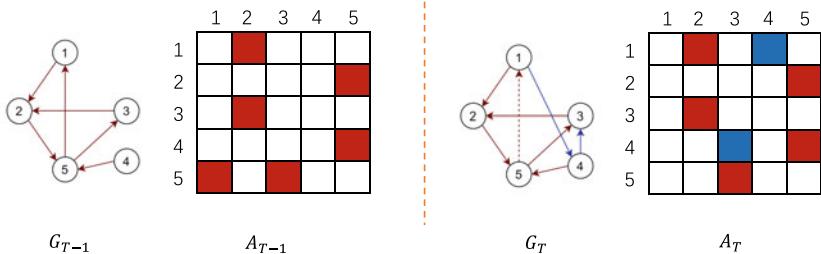


Fig. 20.1 An illustration of network evolution. The structure of the network changes overtime. At time T , $E_{(1,4)}$ and $E_{(4,3)}$ emerge while $E_{(5,1)}$ vanishes, which is reflected in the change of A , with those elements equal to 1 represented by filled squares

Definition 20.2 (Dynamic Network Link Prediction) Given a sequence of graphs with length N , $S=\{G_{t-N}, \dots, G_{t-1}\}$, Dynamic Network Link Prediction (DNLP) aims to learn a function that maps the input sequence S to G_t .

The structure of a dynamic network evolves with time, and as illustrated in Fig. 20.1, links may emerge or disappear, which can be observed through changes in the adjacency matrix over time. The goal is to find the links that are most likely to appear or disappear at the next time span. This can also be mathematically formulated as an optimization problem where a matrix with elements either equal to 0 or 1 is sought to best fit the ground truth.

20.3.2 Our Framework

A novel deep learning model is proposed in this work. The model combines the architecture of encoder–decoder and stacked LSTM with the overall framework shown in Fig. 20.2. The encoder is positioned at the entrance of the model to learn the

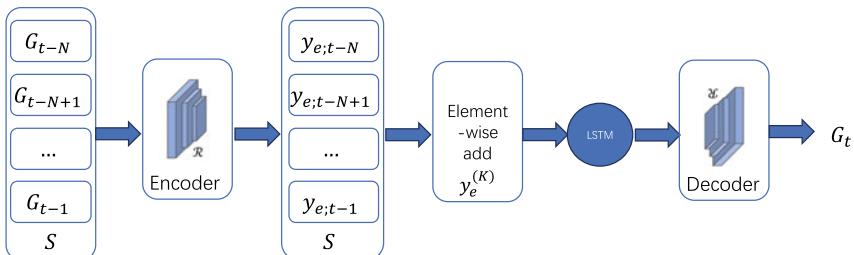


Fig. 20.2 The overall framework of our method. Given a sequence of graphs with length N , $\{G_{t-N}, G_{t-N+1}, \dots, G_{t-1}\}$, the encoder maps them into a lower dimensional latent space. Each graph is transformed into a matrix that represents the structural features. And then the stacked LSTM, composed of multiple LSTM cells, learns network evolution patterns from the extracted features. The decoder projects the received feature maps back to the original space to get G_t . Here, σ in LSTM cells is an activation function and we use sigmoid in this paper

highly non-linear network structures, while the decoder converts the extracted features back to the original space. This encoder–decoder architecture can handle spatial non-linearity and sparsity, and the stacked LSTM between the encoder and decoder can learn temporal dependencies. The well-designed end-to-end model can simultaneously learn both structural and temporal features and perform link prediction in a unified manner.

In this work, we first present the necessary terms and notations that will be frequently used in subsequent sections. These are listed in Table [ref{table:notation}](#). The other notations will be explained along with their corresponding equations. It is worth mentioning that a single LSTM cell can be considered as a layer, in which the terms with subscript f denote the parameters of the forget gate, the terms with subscripts i and C represent the parameters of the input gate, and those with subscript o signify the parameters of the output gate.

20.3.2.1 Encoder–Decoder Architecture

Autoencoders are capable of learning representations of data in an unsupervised manner. In order to incorporate this capability into our model, we place an encoder at the entrance of the network to capture the highly non-linear network structure and a graph reconstructor at the end to transform the latent features back into a matrix of fixed shape. However, unlike autoencoders, which do not require any supervision, our model is supervised, as labeled data (A_t) is used to guide the decoder in constructing matrices that better fit the target distributions. The encoder is composed of multiple non-linear perceptions that project the high-dimensional graph data into a relatively lower dimensional vector space. As a result, the obtained vectors can be used to characterize the local structure of vertices in the network. This process can be characterized as

$$\begin{aligned}\mathbf{y}_{e;i}^{(1)} &= \text{ReLU}(W_e^{(1)} s_i + b_e^{(1)}) \\ \mathbf{y}_{e;i}^{(k)} &= \text{ReLU}(W_e^{(k)} \mathbf{y}_{e;i}^{(k-1)} + b_e^{(k)}) \\ \mathbf{Y}_e^{(k)} &= \sum_{t=1}^{N-1} \mathbf{y}_{e;t}^{(k)},\end{aligned}\tag{20.1}$$

where s_i donates i th graph in the input sequence S . $W_e^{(k)}$ and $b_e^{(k)}$ represent the weight and bias of the k th encoder layer, respectively. And $\mathbf{Y}_e^{(k)}$ is the output of k th encoder layer. For an input sequence, each encoder layer processes every term separately and then concatenates all the activations by element-wise adding. Here, we use $\text{ReLU}(x) = \max(0, x)$ [27] as the activation function for each encoder/decoder layer to accelerate convergence.

The decoder with the mirror structure of the encoder receives the latent features and maps them into the reconstruction space under the supervision of A_t , represented by

Table 20.1 Terms and notations used in the framework

Symbol	Definition
L	Number of LSTM cells
\hat{A}_t	Output of the decoder
H	Output of the stacked LSTM
K	Number of encoder/decoder layers
$W_{f,i,C,o}^{(l)}$	Weight of l th LSTM layer
$b_{f,i,C,o}^{(l)}$	Bias of l th LSTM layer
$Y_e^{(k)}, Y_d^{(k)}$	Output of k th encoder/decoder layer
$W_e^{(k)}, W_d^{(k)}$	Weight of k th encoder/decoder layer
$b_e^{(k)}, b_d^{(k)}$	Bias of k th encoder/decoder layer

$$\begin{aligned} \mathbf{Y}_d^{(1)} &= \text{ReLU}(W_d^{(1)} H + b_d^{(1)}) \\ \mathbf{Y}_d^{(k)} &= \text{ReLU}(W_d^{(k)} \mathbf{Y}_d^{(k-1)} + b_d^{(k)}), \end{aligned} \quad (20.2)$$

where H is generated by the stacked LSTM and represents the features of the target snapshot rather than a sequence of features of all previous snapshots used in the encoder. $W_d^{(k)}$ and $b_d^{(k)}$ represent the weight and bias of the k th in decoder, respectively. And $\mathbf{Y}_d^{(k)}$ is the output of k th decoder layer. Another difference is the last layer of the decoder, or the output layer, uses $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ as the activation function rather than ReLU . And the number of units of the output layer always equals to the number of nodes (Table 20.1).

20.3.2.2 Stacked LSTM

Although the encoder-decoder architecture can handle high non-linearity, it is not capable of capturing time-varying characteristics. LSTM [28], as a special kind of recurrent neural network (RNN), has been introduced to address this issue by learning long-term dependencies. An LSTM consists of three gates: a forget gate, an input gate, and an output gate. The first step is to determine what information needs to be discarded from the previous cell state. This operation is performed by the forget gate, which is defined as

$$f_t = \sigma(W_f \cdot [h_{t-1}, Y_e^{(K)}] + b_f), \quad (20.3)$$

where h_{t-1} represents the output at time $t - 1$ and σ donates the activation function sigmoid . K refers to the number of encoder layers. W_f and b_f are the weight and bias of the forget gate in LSTM, respectively. Then the input gate decides what new information should be added to the cell state. First, a sigmoid layer decides what

information the input contains, i_t , should be updated. Second, a tanh layer generates a vector of candidate state values, \tilde{C}_t , which could be added to the cell state. The combination of i_t and \tilde{C}_t represents the current memory that can be used for updating C_t . The operation is defined as

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, Y_e^{(K)}] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, Y_e^{(K)}] + b_C) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t. \end{aligned} \quad (20.4)$$

W_i and W_C are the weights of the input gate in LSTM. b_i and b_C are corresponding biases. Taking the benefit of the forget gate and the input gate, LSTM cell can not only store long-term memory but also filter out the useless information. The output of LSTM cell is based on C_t and it is controlled by the output gate which decides what information, o_t , should be exported. The process is described as

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, Y_e^{(K)}] + b_o) \\ h_t &= o_t * \tanh(C_t), \end{aligned} \quad (20.5)$$

where W_o and b_o represent the weight and bias of the output gate in LSTM, respectively. A single LSTM cell is capable of learning temporal dependencies, but a chain-like LSTM module, known as the stacked LSTM, is more suitable for processing time-series data. The stacked LSTM consists of multiple LSTM cells that receive input signals in the order of time and pass them through the chain of LSTM cells to produce output signals. This allows the model to capture long-term dependencies in the data and make more accurate predictions. We place the stacked LSTM between the encoder and the decoder to learn the patterns under which the network evolves. After receiving the features extracted at time t , the LSTM module turns them into h_t and then feed h_t back to the model at next training step. It helps the model make use of the remaining information of previous training data. It should be always noticed that the numbers of units in encoder, LSTM cells, and decoder vary when N changes. The larger N , the more units we need in the model.

The encoder at the entrance of the model can reduce the dimensionality of each graph, which helps to keep the computation of the stacked LSTM within a reasonable cost. This stacked LSTM module is designed to efficiently handle temporal and sequential data, making it a valuable supplementary component to the encoder in turn.

20.3.3 *Balanced Training Process*

L_2 distance, often applied in regression, can measure the similarity between two samples. But if we simply use it as loss function in the proposed model, the cost could probably not converge to an expected range or result in overfitting due to the sparsity of the network. There are far more zero elements than non-zero elements in A_t , making the decoder appeal to reconstruct zero elements. To address this sparsity

problem, we should focus more on those existing links rather than nonexistent links in back propagation. We define a new loss function as

$$\begin{aligned}\mathcal{L} &= \sum_{i=1}^n \sum_{j=1}^n (a_{t;i,j} - \hat{a}_{t;i,j}) * p_{i,j} \\ &= \| (A_t - \hat{A}_t) \odot P \|_F^2,\end{aligned}\quad (20.6)$$

where \odot means the Hadamard product. For each training process, $p_{i,j} = 1$ if $a_{t;i,j} = 0$ and $p_{i,j} = \beta > 1$ otherwise. And β is a positive value representing the penalty coefficient. Such penalty matrix exerts more penalty on non-zero elements so that the model could avoid overfitting to a certain extent. And we finally use the mixed loss function

$$\mathcal{L}_{total} = \mathcal{L} + \alpha \mathcal{L}_{reg}, \quad (20.7)$$

where \mathcal{L}_{reg} , defined in Eq. (20.8), is a \mathcal{L}_2 regularizer to prevent the model from overfitting and α is a trade-off parameter.

$$\begin{aligned}\mathcal{L}_{reg} &= \frac{1}{2} \sum_{k=1}^K \left(\|W_e^{(k)}\|_F^2 + \|\hat{W}_d^{(k)}\|_F^2 \right) \\ &\quad + \frac{1}{2} \sum_{l=1}^L \left(\|W_f^{(l)}\|_F^2 + \|W_i^{(l)}\|_F^2 \right. \\ &\quad \left. + \|W_C^{(l)}\|_F^2 + \|W_o^{(l)}\|_F^2 \right).\end{aligned}\quad (20.8)$$

The value of each element in A is either 0 or 1. The output data, however, are not one-hot encoded. They are decimals and could go to infinity or move towards the opposite direction theoretically. In order to get a valid adjacency matrix, we impose a sigmoid function at the output layer and then modify the values to 0 and 1 with 0.5 as the demarcation point. That is, there exists a link between i and j if $\hat{a}_{t;i,j} \geq 0.5$ and there is no link otherwise. To optimize the proposed model, we should first make a forward propagation to obtain the loss and then do backpropagation to update all the parameters. In particular, the key operation is to calculate the partial derivative of $\partial \mathcal{L}_{total} / \partial W_{e,d}^{(k)}$ and $\partial \mathcal{L}_{total} / \partial W_{f,i,C,o}^{(l)}$.

We would like to take the calculation of $\partial \mathcal{L}_{total} / \partial W_e^{(k)}$ for instance. Taking partial derivative with respect to $W_e^{(k)}$ of Eq. (20.7), we have

$$\begin{aligned}\frac{\partial \mathcal{L}_{total}}{\partial W_e^{(k)}} &= \frac{\partial \mathcal{L}}{\partial W_e^{(k)}} + \alpha \frac{\partial \mathcal{L}_{reg}}{\partial W_e^{(k)}} \\ &= \frac{\partial \mathcal{L}}{\partial A_t} \cdot \frac{\partial A_t}{\partial W_e^{(k)}} + \alpha \|W_e^{(k)}\|_F.\end{aligned}\quad (20.9)$$

According to Eq. (20.6), we can easily obtain

$$\frac{\partial \mathcal{L}}{\partial A_t} = 2(A_t - \hat{A}_t) \odot P. \quad (20.10)$$

To calculate $\partial A_t / \partial W_e^{(k)}$, we should iteratively take partial derivative with respect to $\partial W_e^{(k)}$ on both sides of Eq. (20.1). After getting $\partial \mathcal{L}_{total} / \partial W_e^{(k)}$, we update the weight by

$$W_e^{(k)} = W_e^{(k)} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial W_e^{(k)}}, \quad (20.11)$$

where λ is the learning rate which is set as 1e-3 in the following experiments.

As for $\partial A_t / \partial W_d^{(k)}$ and $\partial \mathcal{L}_{total} / \partial W_{f,i,C,o}^{(l)}$, the calculation of partial derivative almost follows the same procedure. The calculation of $\partial A_t / \partial W_d^{(k)}$ goes as follows:

$$\begin{aligned} W_d^{(k)} &= W_d^{(k)} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial W_d^{(k)}} \\ \frac{\partial \mathcal{L}_{total}}{\partial W_d^{(k)}} &= \frac{\partial \mathcal{L}_{total}}{\partial A_t} \cdot \frac{\partial A_t}{\partial W_d^{(k)}} + \alpha \| W_k^{(d)} \|_F \\ \frac{\partial A_t}{\partial W_d^{(k)}} &= H \cdot \prod_{i=0, i \neq k}^K W_d^{(i)}. \end{aligned} \quad (20.12)$$

It is more complicated for $\partial \mathcal{L}_{total} / \partial W_{f,i,C,o}^{(l)}$ because the recurrent network makes use of cell states at every forward propagation cycle and we need to take cell states of two consecutive moments into consideration. We would take $\partial \mathcal{L}_{total} / \partial W_f^{(L)}$ as an example, which is shown below, and the rest are almost the same.

$$\begin{aligned} W_f^{(L)} &= W_f^{(L)} - \lambda \frac{\partial \mathcal{L}_{total}}{\partial W_f^{(L)}} \\ \frac{\partial \mathcal{L}_{total}}{\partial W_f^{(L)}} &= \frac{\partial \mathcal{L}_{total}}{\partial A_t} \cdot \frac{\partial A_t}{\partial H} \cdot \frac{\partial H}{\partial W_f^{(L)}} + \alpha \| W_L^{(f)} \|_F \\ \frac{\partial A_t}{\partial H} &= \prod_{i=1}^K W_d^{(i)}. \end{aligned} \quad (20.13)$$

The calculation of $\partial H / \partial W_f^{(L)}$ needs to recurrently take the partial derivatives of Eq. (20.5). The auto-grad function implemented in deep learning framework, like TensorFlow,¹ would be useful.

¹ <https://tensorflow.google.cn/>.

20.3.4 Complexity Analysis

Suppose our method consists of an encoder layer with m_1 units, an LSTM layer whose hidden size is m_2 and a decoder layer, then the complexity is obtained by

$$\mathcal{O}(n) \sim n(m_1 + m_2) + 4(m_1m_2 + m_2^2 + m_2), \quad (20.14)$$

where n is the number of nodes of G_t .

20.4 Experiments

20.4.1 Datasets

Five real-world dynamic networks, which are all human contact networks, were used for the experiments. In these networks, nodes represent humans and links represent their contacts. The types of contacts can vary from face-to-face proximity to emailing and other forms of communication. Below is a detailed description of these datasets.

- **CONTACT [29]:** It is a human contact dynamic network of face-to-face proximity. The data are collected through the wireless devices carried by people. A link between person s (source) and t (target) emerges along with a timestamp if s gets in touch with t . The data are recorded every 20 s and multiple edges may be shown at the same time if multiple contacts are observed in a given interval.
- **ENRON [30] and RADOSLAW [31]:** They are email networks and each node represents an employee in a mid-sized company. A link occurs every time an email sent from one to another. ENRON records email interactions for nearly 6 months and RADOSLAW lasts for nearly 9 months.
- **FB-FORUM [32]:** The data were attained from a Facebook-like online forum of students at University of California, Irvine, in 2004. It is an online social network where nodes are users and links represent interactions (e.g., messages) between them. The records span more than 5 months.
- **LKML [33]:** The data for this study were obtained from the Linux kernel mailing list. The nodes in the network represent users, identified by their email addresses, and each link represents a reply from one user to another. We focused on a subset of 2210 users who were recorded between January 1, 2007 and April 1, 2007. Subsequently, a dynamic network was constructed using only the links that appeared between these users and other users over a period of 3 years, from April 1, 2007 to December 31, 2013.

Prior to the commencement of training, the datasets were divided into segments at a fixed interval. Each segment constitutes a time window within which the links form a snapshot. To model the dynamic network as a sequence with incremental links is not pursued in this study. Instead, we discard the links that do not reappear in

the subsequent eight intervals, given that connections between individuals are likely to be fleeting. The interval for each dataset may vary depending on its duration; therefore, to ensure that adequate training data is obtained, in our experiments, we transform each dataset into 320 snapshots with varying intervals and set $N = 10$. In this case, $\{G_{t-10}, \dots, G_{t-1}, G_t\}$ is treated as a sample with the first ten snapshots as the input and the last one as the output. As a result, we can get 310 samples in total. We then group the first 230 samples, with t varying from 11 to 240, as the training set, and the rest 80 samples, with t varying from 241 to 320, as the test set.

20.4.2 Baseline Methods

To validate the effectiveness of our method, we compare it with node2vec, as a widely used baseline network embedding method, as well as four state-of-the-art DNLP methods that could handle time dependencies, including Hybrid-TS [34], Temporal Network Embedding (TNE) [35], conditional temporal RBM (ctRBM) [25], gradient boosting decision tree-based temporal RBM (GTRBM) [26], and deep dynamic network embedding (DDNE) [36]. In particular, the five baselines are introduced as follows.

- **node2vec** [37]: Network embedding is a method used to map the nodes of a network from a higher dimensional space to a lower dimensional vector space. The probability of two nodes being connected is higher when their corresponding vectors are located in close proximity in the lower dimensional space. This implies that the nodes are more similar, and hence have a greater chance of being connected.
- **Hybrid-TS** [34]: Hybrid-TS employs a combination of static link prediction methods and time-series link prediction models to perform dynamic network link prediction. This approach leverages the strengths of both approaches to improve the accuracy and efficiency of link prediction in dynamic networks. By combining static and time-series techniques, Hybrid-TS can effectively capture the patterns and dynamics of network links over time, leading to more accurate predictions.
- **TNE** [35]: In this approach, network evolution is modeled as a Markov process and then the matrix factorization technique is utilized to obtain the embedding vector for each node. This method uses the Markov process to capture the temporal dependencies in the network evolution and the matrix factorization algorithm to reduce the dimensionality of the nodes' representations. By doing so, it enables more accurate predictions of future network links based on the current state of the network.
- **ctRBM** [25]: The model is a generative approach based on temporal Restricted Boltzmann Machines (RBM). It begins by generating vector representations for each node based on their connections over time. These representations capture the temporal dependencies in the network. The model then predicts future links by integrating information from its neighbors. This allows it to generate more accurate

predictions of future network linkages, taking into account both the current state of the network and its past behavior.

- **GTRBM [26]:** The model utilizes the strengths of both Temporal Restricted Boltzmann Machines (tRBM) and Gradient Boosting Decision Trees (GBDT) to effectively learn the hidden dynamic patterns in the network. This allows it to capture the complex interactions between nodes over time, leading to more accurate predictions of future network linkages. The combination of tRBM and GBDT also enables the model to handle large-scale networks with high-dimensional data, improving its scalability and efficiency.
- **DDNE [36]:** The model is similar to an autoencoder, but instead of using an encoder–decoder structure, it uses a Gated Recurrent Unit (GRU) as its encoder to read historical information from the network. The GRU reads in the concatenated embeddings of previous network snapshots and then decodes them into the future network structure. This allows the model to capture the temporal dynamics of the network and generate more accurate predictions of future linkages based on this historical information.

When implementing node2vec, we set the dimension of the embedding vector as 80 for CONTACT, ENRON, and RADOSLAW which have less than 500 nodes. And for FB-FORUM and LKML with larger size, we set the dimension as 256. We grid search over {0.5, 1, 1.5, 2} to find the optimal values for hyperparameters p and q , and then use weighted-L2 [37] to obtain the vector $e_i^{(u,v)}$ for each pair of nodes u and v , with each element defined as

$$e_i^{(u,v)} = |u_i - v_i|^2, \quad (20.15)$$

where u_i and v_i are the i th element of embedding vectors of nodes u and v , respectively. For Hybrid-TS, we use node2vec described above to serve as the static link prediction approach and autoregressive integrated moving average model (ARIMA) serves as the time-series prediction model. ARIMA does prediction over link occurrence frequency series $\{x_{tij}\}$ which is defined as

$$x_{tij} = \sum_{k \in V \text{ and } (i,k), (j,k) \in \text{the link set of } S} \log(\text{degree}(k)). \quad (20.16)$$

The parameters of ARIMA, the number of autoregressive terms p_a , the differential order d_a , and the number of moving average terms q_a are searched within {0,1,2,3}, {0,1} and {0,1,2,3}, respectively.

20.4.3 Evaluation Metrics

There are limited evaluation metrics specifically designed for dynamic network link prediction (DNLP). Most of the evaluation metrics commonly used for static link prediction are also employed for DNLP. The Area Under the ROC Curve (AUC) is a common metric used to measure the performance of a dynamic link predictor.

AUC measures the probability that the predictor assigns a higher score to a randomly chosen existing link than a randomly chosen nonexistent one. A predictor with an AUC value closer to 1 is considered more informative. Other metrics, such as precision, Mean Average Precision (MAP), F1-score, and accuracy, evaluate link prediction methods from a binary classification perspective. However, all of these metrics suffer from the sparsity problem in dynamic networks and cannot provide meaningful measurements for dynamic performance. The Area Under the Precision–Recall Curve (PRAUC) [38] developed from AUC is designed to address this issue. However, removed links in near future, which is an essential aspect of DNLP, are not characterized by PR curve, making PRAUC less effective in this case. Junuthula et al. restricted the measurements to only part of node pairs and proposed the Geometric Mean of AUC and PRAUC (GMAUC) [25] for added and removed links, which better reflects the dynamic performance. Li et al. used SumD that counts the differences between the predicted network and the true one, evaluating link prediction methods in a more rigorous way. However, absolute differences could be misleading. For example, two dynamic link predictors both achieve SumD at 5. However, one predictor mispredicts 5 links in 10 while the other mispredicts 5 in 100. It's evident that the latter one performs better than the former one but SumD fails to reveal this.

In our experiments, we choose AUC and GMAUC, and also define a new metric, *Error Rate*, to evaluate our method and other baseline methods.

- **AUC:** If among n independent comparisons, there are n' times that the existing link gets a higher score than the nonexistent link and n'' times they get the same score, then we have

$$\text{AUC} = \frac{n' + 0.5n''}{n}. \quad (20.17)$$

Before calculation, we randomly sample nonexistent links with the same number of existing links to ease the impact of sparsity.

- **GMAUC:** It is a metric specifically designed for measuring the performance of DNLP. It combines PRAUC (the area under the precision–recall curve) and AUC by taking geometric mean of the two quantities, which is defined as

$$\text{GMAUC} = \left(\frac{\text{PRAUC}_{\text{new}} - \frac{L_A}{L_A+L_R}}{1 - \frac{L_A}{L_A+L_R}} \cdot 2(\text{AUC}_{\text{prev}} - 0.5) \right)^{1/2}, \quad (20.18)$$

where L_A and L_R refer to the numbers of added and removed edges, respectively. $\text{PRAUC}_{\text{new}}$ is the PRAUC value calculated among the new links and AUC_{prev} represents the AUC for the observed links.

- **Error Rate:** It is defined as the ratio of the number of mispredicted links, denoted by N_{false} , to the total number of truly existing links, denoted by N_{true} , which is represented by

$$\text{Error Rate} = \frac{N_{false}}{N_{true}}. \quad (20.19)$$

20.4.4 Experimental Results

For each epoch, we feed 10 historical snapshots, $\{G_{t-10}, \dots, G_{t-1}\}$ to our method and infer G_t . And it is the same for implementing the other four DNLP approaches. For the methods that are not able to deal with time dependencies, i.e., node2vec, there are following two typical treatments: (1) only using G_{t-1} to infer G_t [13] or (2) aggregating previous 10 snapshots into a single network and then do link prediction [15, 36]. We choose the former one when implementing node2vec, because the relatively long sequence of historical snapshots here may carry some disturbing information that node2vec cannot handle, leading to even poor performance.

The performance metrics AUC, GMAUC, and Error Rate were compared between our method and the five baseline methods in a dynamic network link prediction task. Since the patterns of network evolution may change over time, the historically trained model may not capture the pattern in the future. To investigate both short-term and long-term prediction performance, we reported the average values of the three performance metrics for the first 20 test samples and all 80 samples. The results are presented in Fig. 20.3. It can be seen that, generally, our method outperforms the baseline methods in most cases for both short-term and long-term prediction. However, it is worth noting that our method performs slightly worse than TNE on LKML when using AUC and GMAUC as the performance metric. This might be because, for a relatively large network, more training samples are needed to improve the performance of our method. In particular, the poor performances obtained by node2vec indicate that the methods designed for static networks are indeed not suitable for dynamic network link prediction. On the contrary, other dynamic link predictors, such as our method, can achieve much better performance due to their dynamic nature.

In the dynamic network link prediction task, we compare the performance metrics AUC, GMAUC, and Error Rate for each predicted snapshot. We find that node2vec can predict much more links than the truly existing ones, leading to relatively large Error Rates. We argue that this might be due to the classification process of the pre-trained linear regression model, which is not suitable for classifying embedding vectors. As presented in Fig. 20.3, the results demonstrate the best performance of our method on dynamic network link prediction (DNLP). TNE performs poorly on Error Rate because it does not specifically fit the distribution of the network as other deep learning-based methods do. The dramatic difference in Error Rate between our method and TNE indicates that this metric is a valuable addition to

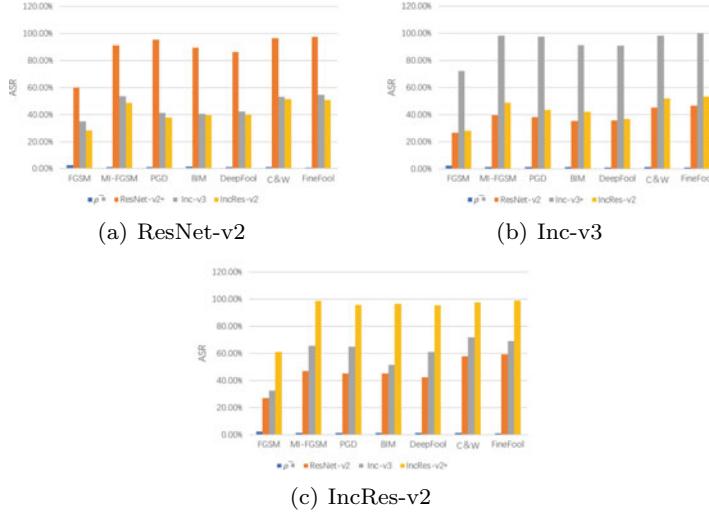


Fig. 20.3 The parameters of our method in the five datasets

AUC in comprehensively measuring the performance of DNLP. Other deep learning-based methods, such as ctRBM and DDNE, have similar performances but could not compete with our method in most cases. It is worth noting that TNE outperforms the others on LKML from the perspective of traditional AUC and GMAUC, which shows its robustness to the scale of networks on these metrics. However, it has much larger Error Rate compared with the other DNLP methods.

For the 80 test samples with $G_{240+\Delta}$ as the output, where Δ varies from 1 to 80, we draw the DNLP performances on the three metrics, obtained by our method, as functions of Δ for the five datasets to see how long it can predict network evolution with satisfying performance. It can be observed from the results that, for RADOSLAW, FB-FORUM, and LKML, the prediction performances are relatively stable, which may be due to their network structures evolving periodically, making it easier to predict snapshots, especially when LSTM is integrated into the deep learning framework. We conducted an analysis to examine the trends of the most commonly used structural properties, including average degree and average clustering coefficient, for the five networks as Δ increased. The purpose of this investigation was to provide further insights into the observed results.

20.5 Conclusion

Our method has been proposed for dynamic network link prediction (DNLP). The model integrates an end-to-end encoder-decoder with a stacked Long Short-Term Memory (LSTM) layer to capture the temporal dependencies between successive net-

work snapshots. By learning both low-dimensional representations and non-linearity, our model is able to better capture the patterns of network evolution. Our experiments on various datasets demonstrate that our model outperforms traditional link prediction methods and achieves state-of-the-art performance. Our method can be utilized in real-life scenarios such as analyzing protein–protein interactions and predicting friendships in social networks. It provides a convenient solution for experts in fields like biology who are not familiar with graph processing by allowing them to feed the model with a sequence of historical snapshots and obtain all the results directly. Additionally, our model can fully utilize both recent and earlier snapshots, making it particularly useful in certain cases such as communication networks.

References

1. Ediger, D., Jiang, K., Riedy, J., Bader, D.A., Corley, C.: Massive social network analysis: mining twitter for social good. In: 2010 39th International Conference on Parallel Processing (ICPP), pp. 583–593. IEEE (2010)
2. Fu, C., Wang, J., Xiang, Y., Wu, Z., Yu, L., Xuan, Q.: Pinning control of clustered complex networks with different size. *Phys. A* **479**, 184–192 (2017)
3. Wang, L., Orchard, J.: Investigating the evolution of a neuroplasticity network for learning. *IEEE Trans. Syst., Man, Cybern.: Syst.* (2018). <https://doi.org/10.1109/TSMC.2017.2755066>
4. Gao, J., Xiao, Y., Liu, J., Liang, W., Chen, C.P.: A survey of communication/networking in smart grids. *Futur. Gener. Comput. Syst.* **28**(2), 391–404 (2012)
5. Kazemilari, M., Djauhari, M.A.: Correlation network analysis for multi-dimensional data in stocks market. *Phys. A* **429**, 62–75 (2015)
6. Ibrahim, N.M.A., Chen, L.: Link prediction in dynamic social networks by integrating different types of information. *Appl. Intell.* **42**(4), 738–750 (2015)
7. Xuan, Q., Fang, H., Fu, C., Filkov, V.: Temporal motifs reveal collaboration patterns in online task-oriented networks. *Phys. Rev. E* **91**(5), 052813 (2015)
8. Xuan, Q., Zhang, Z.Y., Fu, C., Hu, H.X., Filkov, V.: Social synchrony on complex networks. *IEEE Trans. Cybern.* **48**(5), 1420–1431 (2018)
9. Xuan, Q., Zhou, M., Zhang, Z.Y., Fu, C., Xiang, Y., Wu, Z., Filkov, V.: Modern food foraging patterns: Geography and cuisine choices of restaurant patrons on yelp. *IEEE Trans. Comput. Soc. Syst.* **5**(2), 508–517 (2018)
10. Fu, C., Zhao, M., Fan, L., Chen, X., Chen, J., Wu, Z., Xia, Y., Xuan, Q.: Link weight prediction using supervised learning methods and its application to yelp layered network. *IEEE Trans. Knowl. Data Engin* (2018)
11. Lentz, H.H., Koher, A., Hövel, P., Gethmann, J., Sauter-Louis, C., Selhorst, T., Conraths, F.J.: Disease spread through animal movements: a static and temporal network analysis of pig trade in germany. *PLoS ONE* **11**(5), e0155196 (2016)
12. Theocharidis, A., Van Dongen, S., Enright, A.J., Freeman, T.C.: Network visualization and analysis of gene expression data using biolayout express 3d. *Nat. Protoc.* **4**(10), 1535 (2009)
13. Yao, L., Wang, L., Pan, L., Yao, K.: Link prediction based on common-neighbors for dynamic social network. *Proc. Comput. Sci.* **83**, 82–89 (2016). <https://doi.org/10.1016/j.procs.2016.04.102>
14. Ahmed, N.M., Chen, L.: An efficient algorithm for link prediction in temporal uncertain social networks. *Inf. Sci.* **331**, 120–136 (2016)
15. Nguyen, G.H., Lee, J.B., Rossi, R.A., Ahmed, N.K., Koh, E., Kim, S.: Continuous-time dynamic network embeddings. In: 3rd International Workshop on Learning Representations for Big Networks (WWW BigNet) (2018)

16. Xuan, Q., Fang, B., Liu, Y., Wang, J., Zhang, J., Zheng, Y., Bao, G.: Automatic pearl classification machine based on a multistream convolutional neural network. *IEEE Trans. Industr. Electron.* **65**(8), 6538–6547 (2018)
17. Xuan, Q., Xiao, H., Fu, C., Liu, Y.: Evolving convolutional neural network and its application in fine-grained visual categorization. *IEEE Access* (2018)
18. Ozcan, A., Oguducu, S.G.: Link prediction in evolving heterogeneous networks using the narx neural networks. *Knowl. Inf. Syst.* **55**(2), 333–360 (2018)
19. Ozcan, A., Oguducu, S.G., et al.: Multivariate time series link prediction for evolving heterogeneous network. *Int. J. Inf. Technol. Decis. Making (IJITDM)* **18**(01), 241–286 (2019)
20. Ahmed, N.M., Chen, L., Wang, Y., Li, B., Li, Y., Liu, W.: Sampling-based algorithm for link prediction in temporal networks. *Inf. Sci.* **374**, 1–14 (2016)
21. Yu, W., Cheng, W., Aggarwal, C.C., Zhang, K., Chen, H., Wang, W.: Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2672–2681. ACM (2018)
22. Brand, M.: Fast low-rank modifications of the thin singular value decomposition. *Linear Algebra Appl.* **415**(1), 20–30 (2006)
23. Zhang, Z., Cui, P., Pei, J., Wang, X., Zhu, W.: Timers: Error-bounded svd restart on dynamic networks. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
24. Ma, X., Sun, P., Wang, Y.: Graph regularized nonnegative matrix factorization for temporal link prediction in dynamic networks. *Phys. A* **496**, 121–136 (2018)
25. Li, X., Du, N., Li, H., Li, K., Gao, J., Zhang, A.: A deep learning approach to link prediction in dynamic networks. In: Proceedings of the 2014 SIAM International Conference on Data Mining, pp. 289–297. SIAM (2014)
26. Li, T., Wang, B., Jiang, Y., Zhang, Y., Yan, Y.: Restricted boltzmann machine-based approaches for link prediction in dynamic networks. *IEEE Access* (2018)
27. Xu, B., Wang, N., Chen, T., Li, M.: Empirical evaluation of rectified activations in convolutional network. arXiv preprint [arXiv:1505.00853](https://arxiv.org/abs/1505.00853) (2015)
28. Gers, F.A., Schmidhuber, J., Cummins, F.: Neural computation. *Appl. Intell.* (1999)
29. Haggle network dataset – KONECT (2017). <http://konect.uni-koblenz.de/networks/contact>
30. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015). <http://networkrepository.com>
31. Michalski, R., Palus, S., Kazienko, P.: Matching organizational structure and social network extracted from email communication. In: Lecture Notes in Business Information Processing, vol. 87, pp. 197–206. Springer, Berlin (2011)
32. Facebook wall posts network dataset – KONECT (2017). <http://konect.uni-koblenz.de/networks/facebook-wosn-wall>
33. Linux kernel mailing list replies network dataset – KONECT (2017). <http://konect.uni-koblenz.de/networks/lkml-reply>
34. Huang, Z., Lin, D.K.: The time-series link prediction problem with applications in communication surveillance. *INFORMS J. Comput.* **21**(2), 286–303 (2009)
35. Zhu, L., Guo, D., Yin, J., Ver Steeg, G., Galstyan, A.: Scalable temporal latent space inference for link prediction in dynamic social networks. *IEEE Trans. Knowl. Data Eng.* **28**(10), 2765–2777 (2016)
36. Li, T., Zhang, J., Philip, S.Y., Zhang, Y., Yan, Y.: Deep dynamic network embedding for link prediction. *IEEE Access* (2018)
37. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 855–864. ACM (2016)
38. Yang, Y., Lichtenwalter, R.N., Chawla, N.V.: Evaluating link prediction methods. *Knowl. Inf. Syst.* **45**(3), 751–782 (2015)