Security Viewpoints on Explainable Machine Learning

vorgelegt von

M.Sc.
Alexander Warnecke
ORCID: 0009-0006-3617-3968

an der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender: Prof. Dr. Wojciech Samek
Gutachter: Prof. Dr. Konrad Rieck
Gutachter: Prof. Dr. Giorgio Giacinto
Gutachter: Prof. Dr. Pavel Laskov

Tag der wissenschaftlichen Aussprache: 16. April 2024

Berlin 2024

Dedicated to my grandfather Otto.

Abstract

In the swiftly evolving research field of machine learning, there is a growing demand for ensuring both the interpretability and security of algorithms. Recently, a variety of approaches to explain how models arrive at their predictions have been developed and constitute a core component towards enabling trustworthy machine learning. In this thesis, we adopt the viewpoints of a developer, an operator and an adversary when investigating explanations and derive research questions that emerge from each perspective.

From the perspective of a developer leveraging machine learning for security tasks such as malware and vulnerability detection, we elucidate the question of selecting an optimal explainable learning algorithm. Recognizing the need for a coherent measure of comparison, we introduce a set of multiple evaluation metrics to contrast explanations. These metrics are employed on four security tasks spanning diverse neural network architectures. Furthermore, we present an algorithm to extract meaningful explanations, offering a compact lens into the model and dataset for further investigation. During the analysis of these prototypical explanations, we unveil a pressing concern that many existing models do not necessarily address the issues they were initially trained for.

Next, we transition to the viewpoint of an operator of a machine learning service and show how explanations can reveal privacy leaks in machine learning models. Our findings underline the need for strategies for data adjustments without significant disruption to the model parameters, especially in the light of legislature like the General Data Protection Regulation (GDPR). In this endeavor, we introduce the concept of certified unlearning for features and labels with a framework based on approximate parameter updates. Conditions that guarantee certified unlearning are derived and tested across a variety of datasets and tasks.

Lastly, adopting the vantage point of a potential adversary, we present innovative attack mechanisms exploiting model weaknesses unveiled by explanation techniques. Firstly, we introduce dormant minimal backdoors, a novel approach to embed stealthy backdoors bypassing standard detection paradigms by manipulating hardware accelerators. Secondly, using the similarity of edge detection and explanations for image classifiers, we present the concept of model-independent adversarial examples. These adversarial examples sidestep the need for access to model parameters or black-box queries, confining their operation exclusively to the input presented to the model.

Collectively, this work enlightens about chances and challenges when adapting explanations into machine learning based systems. On the one hand, explanations can effectively support practitioners in evaluating and debugging datasets and models. On the other hand, problems like information leakage and attack detection must be addressed and understood further to pave the way to trustworthy and robust learning systems.

Zusammenfassung

Im schnelllebigen Forschungsfeld des maschinellen Lernens besteht eine wachsende Nachfrage nach der Gewährleistung von Interpretierbarkeit und Sicherheit der eingesetzten Algorithmen. In letzter Zeit wurden verschiedene Ansätze zur Erklärung der Funktionsweise von Modellen entwickelt, die einen Kernbestandteil zur Ermöglichung von vertrauenswürdiger künstlicher Intelligenz darstellen. In dieser Arbeit nehmen wir die Perspektiven eines Entwicklers, eines Betreibers und eines Angreifers ein, wenn wir Erklärungen betrachten und leiten Forschungsfragen ab, die aus jeder Perspektive entstehen.

Aus der Sicht eines Entwicklers, der maschinelles Lernen für Aufgaben wie Schadsoftware- und Schwachstellenerkennung nutzt, betrachten wir die Frage der Auswahl eines optimalen Erklärverfahrens. Um die Notwendigkeit eines kohärenten Vergleichsmaßes zu erkennen, stellen wir mehrere Bewertungsmetriken vor, um Erklärungen zu vergleichen. Diese Metriken werden auf vier Datensätzen unter Verwendung verschiedener neuronaler Netzwerkarchitekturen evaluiert. Außerdem präsentieren wir einen Algorithmus zur Auswahl interessanter Erklärungen, der einen kompakten Einblick in das Modell und den Datensatz für weitere Untersuchungen bietet. Die Analyse dieser prototypischen Erklärungen zeigt, dass viele Modelle nicht unbedingt die Probleme lösen, für die sie ursprünglich trainiert wurden.

In der Perspektive eines Betreibers eines maschinellen Lernservices zeigen wir, wie Erklärungen Datenlecks in maschinellen Lernmodellen aufdecken können. Unsere Ergebnisse verdeutlichen die Notwendigkeit von Ansätzen zur Datenanpassung ohne dabei die Modellparameter stark zu verändern, insbesondere vor dem Hintergrund von Gesetzen wie der Datenschutz-Grundsatzverordnung (DSGVO). Zu diesem Zweck führen wir das Konzept des „zertifizierten Verlernens" für Merkmale und Labels ein, das auf approximativen Parameteraktualisierungen basiert. Bedingungen, die "zertifiziertes Verlernen" garantieren, werden abgeleitet und in mehreren Datensätzen und Aufgaben getestet.

Zuletzt nehmen wir den Standpunkt eines Angreifers ein und präsentieren neuartige Angriffsmechanismen, die von Erklärungstechniken aufgedeckte Modellschwächen ausnutzen. Zunächst führen wir „schlafende minimale Hintertüren" ein, einen neuartigen Ansatz, um Hintertüren in Lernmodelle einzufügen, indem Hardwarebeschleuniger manipuliert werden. Außerdem präsentieren wir einen neuen Angriff, um durch minimale Eingabeveränderungen Bildklassifikatoren in die Irre zu leiten, welcher auf der Ähnlichkeit von Erklärungen und Kantendetektionsalgorithmen beruht. Der vorgestellte Angriff benötigt weder Zugang zu den Modellparametern noch mehrere Anfragen an das Lernmodell und operiert daher ausschließlich auf der dem Modell präsentierten Eingabe.

Insgesamt diskutiert diese Arbeit Chancen und Herausforderungen für den Einsatz von Erklärungen im maschinenllen Lernen. Einerseits können Erklärungen Anwender effektiv bei der Bewertung und Fehlerbehebung von Datensätzen und Modellen unterstützen. Andererseits müssen Probleme wie Informationslecks und Angriffserkennung weiter untersucht werden, um den Weg zu vertrauenswürdigen und robusten Lernsystemen zu ebnen.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor Prof. Dr. Konrad Rieck for awakening my interest in machine learning in Göttingen and giving me the chance to do a PhD. Thank you for believing in my ideas, all the discussions about them and for cheering me up in the difficult phases every PhD student has to go through.

Moreover, I would like to thank my two co-referees Prof. Dr. Giorgio Giacinto and Prof. Dr. Pavel Laskov for reading this thesis. I know that you are busy and have full schedules, I appreciate that you take your time for me.

I'm glad to have received funding by the German Research foundation under Germany's Excellence Strategy EXC 2092 CASA-390781972 and by the German Federal Ministry of Education and Research under the grants BIFOLD-23B and VAMOS-16KIS0534.

I'm also extremely grateful for all the people I met and worked with. Firstly, thanks to all members of the MLSEC research group for all the fun, discussions and research projects I was having with you. Special thanks to Prof. Dr. Christian Wressnegger for introducing me to the system security circus and teaching me how academia works. I would like to acknowledge the contributions of all my co-authors to my work, especially Dr. Daniel Arp, Lukas Pirch, Julian Speith and Jan-Niklas Möller. Last but not least, thank you Frank Rust and Stefan Czybik for setting up amazing servers and restarting them when required. Thank you Katja Barkowsky and Sarah Hashmi for handling the bureaucracy for me.

Life is easier when you have good friends, therefore I would like to thank Tobias Meier, Janis Schaab and Roman Wohlgemuth for always being there and for lots of great discussions about life and other things. The "bosscast" we started during the Covid pandemic was the best idea we've ever had. I would also like to thank the LG10 carnival group for providing me with a refreshing viewpoint on the life outside of academia, for the great sleepover parties at the lake and father's day hikes through the forests around Lauenberg. It is great to have a place where you can always come and feel welcome and home.

Above all, I would like to thank my family for their support throughout my life. My grandparents, Karin and Otto, deserve special thanks for taking care of me after school. I know that it wasn't always easy for you. To my mother Martina and my sister Esther, thank you for always being there and encouraging me to pursue the things that bring me happiness. Nadine, my sister-in-law, brought joy and laughter into my life. I appreciate the wonderful memories we shared and the fantastic birthday cakes you made for me. Philine, thank you for showing me the world through your wonderful, unbiased eyes and for bringing so much joy into my life. To my wife Sandra, thank you for your unconditional support and for always believing in me. I love you dearly.

Publications

This thesis contains ideas and results that have been published by the author in the following papers aswell as currently unpublished concepts in the field of adversarial machine learning. Publications listed with a filled square (■) have been authored by the thesis author and those with an empty square (□) originate from collaborations under the lead of other researchers.

- ■ Alexander Warnecke, Daniel Arp, Christian Wressnegger and Konrad Rieck. Evaluating Explanation Methods for Deep Learning in Security. In *Proc. of the 5th IEEE European Symposium on Security and Privacy (EuroS&P)*. 2020.

- □ Lukas Pirch*, Alexander Warnecke*, Christian Wressnegger and Konrad Rieck. TagVet: Vetting Malware Tags using Explainable Machine Learning. In *Proc. of the 14th ACM European Workshop on Systems Security (EuroSec)*. 2021.

- □ Tom Ganz, Martin Härterich, Alexander Warnecke and Konrad Rieck. Explaining Graph Neural Networks for Vulnerability Discovery. In *Proc. of the 14th ACM Workshop on Artificial Intelligence and Security (AISEC)*. 2021. **Best paper award.**

- □ Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro and Konrad Rieck. Dos and Don'ts of Machine Learning in Computer Security. In *Proc. of the 31st USENIX Security Symposium*. 2022. **Distinguished paper award.**

- ■ Alexander Warnecke, Lukas Pirch, Christian Wressnegger and Konrad Rieck. Machine Unlearning of Features and Labels. In *Proc. of the 30th Network and Distributed System Security Symposium (NDSS)*. 2023.

- ■ Alexander Warnecke*, Julian Speith*, Jan-Niklas Möller, Konrad Rieck and Christof Paar. Evil from Within: Machine Learning Backdoors through Hardware Trojans. *Technical report, arXiv:2304.08411*. 2023.

*Authors contributed equally

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

In the last decade, the field of machine learning has experienced an unprecedented surge, transforming the landscape of various industries and research domains. From image recognition [156] and natural language processing [268] to medical diagnosis [88] and computer security [19], machine learning techniques have demonstrated remarkable capabilities in solving complex problems. This revolution is owed to the rise of deep learning architectures, availability of vast datasets, and computational power that have driven machine learning to new heights of accuracy and applicability.

However, this impressive progress has been accompanied by a critical challenge: the lack of transparency and interpretability in the decision-making processes of machine learning models. As machine learning systems become integral to high-stakes applications such as autonomous vehicles [70] or intrusion detection in critical infrastructures [259], the importance of understanding how these models arrive at their decisions has never been more pronounced. The black-box nature of many machine learning algorithms has raised concerns about their reliability, accountability, and potential biases contained in them, necessitating the development of algorithms to explain their predictions. Such clarity is crucial in fostering societal trust in machine learning, particularly for operations in sensitive domains.

Explainable Machine Learning is a novel subfield that aims to bridge the gap between the remarkable predictive power of complex models and the human need for comprehensible and justifiable decisions. The pursuit of explainability is not just a philosophical endeavor; it has real-world implications for issues like legal compliance, ethical considerations, and public trust. The need to strike a balance between accuracy and transparency thus extends the classical machine learning pipeline of training a model on a dataset and measuring its performance on unseen data by an algorithm that generates explanations for inputs fed to the model. Analyzing these explanations the practitioner can draw conclusions about the data and model used and gather insights about the learning task at hand.

Although different approaches for generating explanations have been developed in the last decade [e.g. 25, 266], the majority of them computes *relevance-* or *saliency* scores for each feature of an input to indicate its importance for the classification of the model at hand. Utilizing these scores for analysis has quickly caught the attention of security researchers since they can guide an analyst to weak spots of models or reveal sensitive information from the training data that is used at inference time. In the last decade, a plethora of vulnerabilities that compromise model operation have been found at different stages of the machine learning pipeline [e.g. 205]. Exploiting these attack vectors and adjusting models to prevent them will hopefully pave the way to trustworthy machine learning in the long run. In the following, we present three examples that show how explainable machine learning can be a valuable tool to make learning models more secure and thereby derive the research questions that will be discussed throughout the thesis.

Firstly, let us consider a learning model to detect vulnerabilities in source code. Such vulnerabilities pose a severe threat to modern software but detecting them manually is a tedious and error-prone work. Therefore, approaches for automatic detection of such vulnerabilities have shown promising results in recent years [166, 310, 311]. The code snippet in Figure 1.1 (top) contains a critical call to the `strcpy` function and three different explanations (below) obtained using different algorithms. The underlying model is a recurrent neural network [166] that processes code *token-wise*, thus treating it like natural language and thereby achieves an excellent detection performance. If a token is a variable name or a function, for example, its name will be replaced by a general identifier like `VAR1` as shown in the explanations below. Investigating the given explanations, we see that the first method provides a nuanced representation of the relevant features whereas the second method generated an unsharp explanation due to a lack of sparsity. The third approach provides an explanations that even contradicts the first one. Note that the variables `VAR2` and `VAR3` receive a positive relevance (orange) in the first case but a negative relevance (blue) in the third one.

```
c = split(arg[i],"=",&n);
block_flgs = strcpy((xmalloc(strlen(c[1]) + 1)),c[1]);
```

```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) + INT0 ) ) ,
    VAR1 [ INT0 ] ) ;
```

```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) + INT0 ) ) ,
    VAR1 [ INT0 ] ) ;
```

```
VAR0 = FUN0 ( VAR1 [ VAR2 ] , STR0 , & VAR3 ) ;
VAR0 = strcpy ( ( FUN0 ( strlen ( VAR1 [ INT0 ] ) + INT0 ) ) ,
    VAR1 [ INT0 ] ) ;
```

Figure 1.1: Explanations for the prediction of the security system VulDeePecker on a code snippet from the original training dataset.

The fact that each of the explanation methods is built upon sound assumptions that justify its usage confronts the practitioner with different questions: Which explanation method should I use for the learning problem at hand and how can I compare different methods? We aim for an answer by proposing different metrics for comparing and evaluating explanation methods. Concretely, we focus on different datasets and models from computer security applications that pose special requirements to the explanation methods and thus deserve an own special treatment. We will see that certain approaches are indeed generating better explanations than other ones and should thus be preferred in practical applications.

Next, we investigate models for classifying e-mails into the categories "spam" (malicious) or "ham" (benign). In today's interconnected digital landscape, the convenience of email communication has brought about numerous benefits for both individuals and businesses. However, alongside the advantages, a persistent harm has emerged in the form of spam emails. These malicious messages often carry dangerous payloads, including malware, ransomware, and phishing attacks. If successful, such attacks can compromise sensitive company data, disrupt operations, and lead to financial losses. Machine learning based detectors for spam mails are an effective tool since a lot of e-mail traffic is readily available for training in companies and the performance of such models is remarkably high [24]. The majority of spam mail detectors transform the e-mail content to a "bag of words", indicating which words are present, and use the frequencies of occurrence in spam- and ham-mails for classification. Figure 1.2 shows four example e-mails, two from each class, with the relevance of the five most important words highlighted according to their score. The e-mails originate from a large corpus of e-mail traffic from the american company *Enron* [183].

Id	Spam E-Mail	Spam E-Mail	Ham E-Mail	Ham E-Mail
0	online	remove	2000	enron
1	money	prices	love	gas
2	security	http	deal	2000
3	read	removed	houston	hpl
4	million	80	bob	teco

Figure 1.2: Different explanations from a machine learning based spam detector. For each e-mail the five most important words are highlighted.

Analyzing the token relevance scores, we find that important words in the spam mails are indeed linked to phishing attempts using financial baits (”money”, ”million”, ”http”). However, we also notice the presence of seemingly irrelevant features like dates (”2000”), content from signatures (”enron”, ”houston”) or even the names of employees (”bob”). Such features that are disconnected from the actual learning task but still important can be considered as *artifacts* from the learning problems that are problematic in various ways: Firstly, evading the classifier can be performed easily for spam mail authors by including these artifacts in the crafted e-mails. Secondly, recently introduced legislation, like the General Data Protection Regulation (GDPR) [1] in the European Union include provisions that require the *right to be forgotten*. This mandates companies to take *reasonable steps* to achieve *the erasure of personal data concerning [the individual]* [1]. Depending on the dataset that was used to train the classifier, individuals can claim their right to be forgotten and enforce the model owner to delete their data. However, once a machine learning model is trained it is difficult to remove information from the training data, a process recently named *machine unlearning* [38]. While explainable machine learning can uncover potential artifacts related to personal data, a mechanism for removal – at the best case without retraining the model from scratch – is required. In this thesis we will propose such a method to allow machine unlearning of features and labels by closed-form model updates and derive theoretical guarantees that allow for *certified unlearning*.

Finally, we consider a machine learning models for image classification, being one of the research fields that attracts the most attention in these days. Figure 1.3 shows different datapoints that were used to train a neural network to detect German traffic signs (top) along with the model classfication (column title) and the corresponding explanation (bottom). We can see that the model oftentimes uses only the shapes of the traffic sign to derive the classifcation and only in the case of the roundabout leverages information from inside the traffic sign. Surprisingly, we notice that the rightmost image depicts a stop sign which is classified incorrectly and where all relevance is concentrated in a small region of the image. This indicates

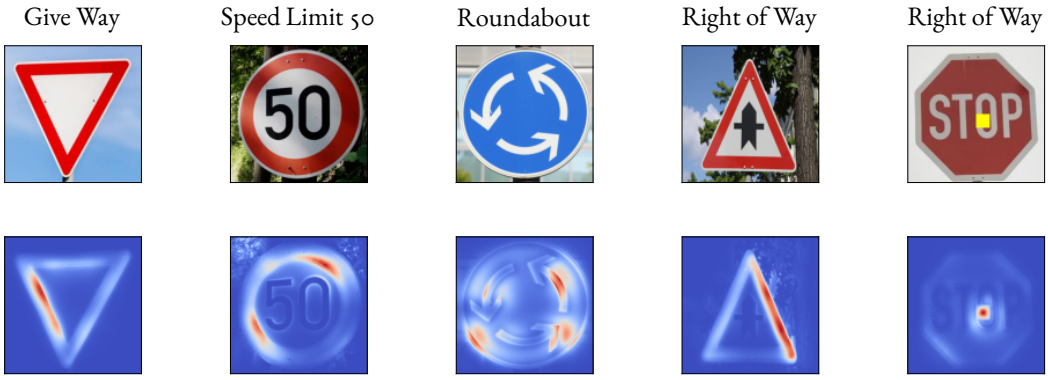


Figure 1.3: Saliency maps for a deep neural network trained for traffic sign recognition. The top row shows training images and the network prediction is above them. Red color in the saliency map indicates high importance whereas blue color indicates irrelevance.

that a *backdooring attack* [107, 170, 242] has been performed on the underlying machine learning model during the training process by an adversary. Such a backdoor refers to a subtle and strategically embedded trigger or pattern within the model that, when appropriately activated, can cause the performance to degrade or produce erroneous results. Backdooring attacks target the model's architecture itself, lying dormant until triggered by specific input conditions, thus evading detection during regular training and testing phases. These maliciously implanted vulnerabilities can compromise the integrity and reliability of machine learning systems, leading to catastrophic consequences in safety-critical applications. For instance, a self-driving car that relies on the neural network for street sign recognition from Figure 1.3 endangers the lives of passengers on the road and undermines the public trust in this new technology.

For an effective explanation method, it is paramount to highlight the trigger in Figure 1.3, thereby unmasking the maliciously injected behavior. Indeed, there exist multiple approaches that leverage saliency maps to detect backdoors in learning models automatically [e.g. 76, 128]. From the perspective of an adversary, the goal then shifts to embedding a backdoor that is invisible for explanation methods in order to avoid detection by these methods. In response to this challenge, this thesis presents a novel form of backdoors called *minimal dormant backdoors* that exploit the fact that the hardware executing the model can be compromised. These backdoors, once installed, allow execution of trojans at the inference step and thus remain undectable for detection methods that exist at the time of this writing. To be applicable in realistic setups, the number of model parameter changes is minimized and we will see that as few as three parameter changes are sufficient to implement effective backdoors into modern neural network architectures.

1.2 Contributions

In this thesis we present *security viewpoints* on explainable machine learning, offering insights from distinct vantage points and addressing questions that stem from these perspectives. The initial standpoint caters to machine learning practitioners who seek to leverage explanations for comprehending the model's learning process or for engaging in subsequent analytical steps to extract novel insights. The second viewpoint zeroes in on the challenges concerning privacy protection that emerge when implementing explanation methods in real-world scenarios. Finally, we slip into the role of an adversary aiming to attack the functionality of a machine learning model using the insights gained from explainations. Notably, this work encompasses four contributions:

- *Evaluating Explanation Methods.* Practitioners have to choose explanation methods, however it is unclear how they can be compared and which one is suited best for an application at hand. We propose six evaluation criteria for comparison covering general concepts as well as those relevant for computer security applications. In an extensive study with different neural networks and datasets, we compare different explanation approaches to highlight advantages and disadvantages when applied in practice.

- *From explanations to insights.* Employing explainable machine learning proves to be a valuable asset across numerous security applications. To ease analysis, we propose a scheme that allows navigating through the space of explanations by using *prototypes* and *outliers*. In diverse domains, spanning from Android malware detection to vulnerability assessment and dynamic malware analysis, we demonstrate the capacity of our approach to elevate the depth of analysis and offer fresh perspectives to analysts.

- *From explanations to unlearning.* Once sensitive features or wrongly labeled data has been detected, it is a challenging task to remove it from the model *after training*. Since modern machine learning models are complex and take long to train we propose the first method to efficiently remove features and labels using a closed form update on the model parameters. We derive conditions under which *certified unlearning*, a strong theoretical guarantee, can be obtained when using our updates in practice and evaluate them on different datasets and machine learning models.

- *From explanations to attacks* Recent research has shown that modern neural networks suffer from adversarial examples and that backdoors can be inserted into them when controlling a small fraction of the training data. We derive the concept of *minimal*

dormant backdoors, a new class of neural network trojans that circumvent detection methods by using compromised hardware and lying dormant until configured by an adversary. Moreover, motivated by explanations for image classifiers, we also propose *model independent adversarial examples* based on edge detection algorithms. Crafting these perturbations requires neither parameter access nor model queries and still allows fooling modern networks efficiently.

1.3 Structure of this Thesis

Chapter 2 presents the fundamentals of machine learning, explanation techniques and attacks on neural networks that are required to follow the remainder of this thesis.

Chapter 3 introduces evaluation methods for explanation methods and a large scale study on datasets and models for computer security applications. The results have been published at the *IEEE European Symposium on Security and Privacy 2020* and the *ACM Workshop on Artificial Intelligence and Security 2021* and provide the practitioner with guidance which explanation methods to use in practice.

Chapter 4 presents insights on different security datasets and models using explainable machine learning. We show how decompiled android applications can be compressed with respect to their behavior and how malware tags can be vetted using dynamical malware analysis. The insights have been published at the *USENIX Security Symposium 2022* and the *ACM Workshop on Systems Security 2021*.

Chapter 5 proposes a framework to unlearn features and labels from machine learning models and derives a theoretical foundation for unlearning in this setting. The approaches are evaluated on different models and datasets and analyzed theoretically. The results have been published at the *Network and Distributed System Security Symposium 2023*.

Chapter 6 discusses new attacks on machine learning models, namely a minimal programmable backdoor for neural networks that requires very few parameter changes to be activated and a new class of adversarial examples that require neither access to the parameters nor queries to the model. At the time of this writing these results have not been published at peer-reviewed conferences yet.

2 Background

In the following chapter we present the theoretical background required to follow the remaining concepts and problems discussed in this thesis. Firstly, we introduce supervised machine learning problems and optimization methods to solve them. Regarding learning models, we will mostly focus on neural networks since they are the dominating models that appear throughout this thesis. Secondly, we present explainable machine learning algorithms that allow to trace back decisions of neural networks in different ways. Finally, adversarial examples, backdoor attacks and attacks on explainable learning techniques are introduced as important threats to modern neural networks.

2.1 Machine Learning

Machine learning is an interdisciplinary research field at the intersection of computer science, statistics, and optimization, focused on developing algorithms and models that enable computers to learn patterns autonomously from datasets. Thereby, these algorithms circumvent the need for explicit programming to solve complex problems across domains such as image and speech recognition and natural language processing. These algorithms can be broadly categorized into three types depending on the way the training data to extract insights from is available, namely unsupervised learning, supervised learning and reinforcement learning.

In this thesis, we will deal with models that have been trained in a supervised fashion, which describes a paradigm where models are trained using *labeled* datasets, i.e., input data is paired with corresponding desired outputs. The learning process aims to make the model learn the underlying relationships between inputs and outputs, enabling it to make accurate predictions or classifications for new, unseen data instances. This approach hinges on the availability of well-annotated training data, allowing the model to iteratively refine its understanding of the data's characteristics and generalize to novel examples.

Learning Problem In the following, we introduce a mathematical notation for supervised learning problems, where we mostly rely on the books of Bishop [36] and Smola and Vishwanathan [257]. In general, we are given a dataset $D = \{z_1, \dots, z_n\}$ with each point $z_i = (x, y)$ consisting of a datapoint $x \in \mathcal{X}$ and a label $y \in \mathcal{Y}$ in a supervised learning setup [257]. We assume that the *feature space* $\mathcal{X} = \mathbb{R}^d$ and therefore use a bold face notation for vectors $\mathbf{x} \in \mathcal{X}$ from now on whereas the j-th entry of $\mathbf{x}$ is denoted by x_j. The structure of the label set $\mathcal{Y}$ depends on the learning task at hand: When aiming to predict a numerical value, like the price of a stock at a given time given a history of prices, the label will be a positive real number, i.e., $\mathcal{Y} = \mathbb{R}_+$. Such problems are also called *regression problems* [36]. The dominating task for this thesis, however, is called *classification* and in this case $\mathcal{Y}$ constitutes a set of C *classes* into which the points in $\mathcal{X}$ can be categorized. Thus, we denote an element in $\mathcal{Y}$ either by the class $y \in \{1, \dots, C\}$ to which $\mathbf{x}$ belongs or by a vector $\mathbf{y}$ representing a discrete probability distribution over the classes, i.e., all entries sum up to one and the entry y_k indicates the probability that $\mathbf{x}$ belongs to class k. At the first glance, it might seem like a strong requirement that the outputs of a learning model form a probability distribution over the class labels. However, such a distribution can be obtained relatively easy for an output vector $\mathbf{y}$ by applying the *softmax* transformation $\hat{y}_i = \exp(y_i) / \sum_k \exp(y_k)$ to every entry y_i of $\mathbf{y}$. A special softmax distribution is called *one-hot* encoding, where only one entry in $\mathbf{y}$ is set to one, oftentimes to indicate a ground truth for training labels [36].

A *learning model* is a mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ that can predict labels for input instances $\mathbf{x} \in \mathcal{X}$ [103]. The prediction depends on the *parameters* $\theta \in \Theta$ of the learning model, where the parameter space $\Theta \subset \mathbb{R}^m$ is the set of all possible parameter vectors. The process of finding "good" parameters θ is called *training* and is performed by optimizing a *loss function* $\ell : \mathcal{X} \times \mathcal{Y} \times \Theta \to \mathbb{R}$ that measures the difference between the prediction of f_θ for a datapoint $\mathbf{x}$ and its true label $\mathbf{y}$. Loss functions are often defined by the notation $\ell : \mathcal{Y} \times \mathcal{Y}$, however for us it is advantageous to use a notation involving the learning parameters as used by Smola and Vishwanathan [257]. The larger the difference between the label and the model prediction,

the larger the value of ℓ will be, thus we aim to minimize the loss function for our dataset D. If we assume that the occurrence of a datapoint in $\mathcal{X}$ and its correct label can be characterized by a density function $p(\mathbf{x}, \mathbf{y})$ we can define an optimal set of parameters by minimizing the expected loss $R_\infty(f_\theta)$ with respect to θ [36], where

$$R_\infty(f_\theta) = \int_{X\times Y} \ell(\mathbf{x}, \mathbf{y}, \theta) dp(\mathbf{x}, \mathbf{y}).$$

In practice, we usually don't know the underlying distribution $p(\mathbf{x}, \mathbf{y})$ and therefore assume that the datapoints in our dataset D have been drawn independently and are distributed identically according to $p(\mathbf{x}, \mathbf{y})$ [36]. Under this assumption we can define the optimal set of parameters θ^* such that the *empirical risk* $R_D(f_\theta)$ is minimized,

$$\theta^* = \underset{\theta\in\Theta}{\operatorname{argmin}}\ R_D(f_\theta) = \underset{\theta\in\Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(z_i, \theta). \tag{2.1}$$

Example 1 *Linear Regression*

As a first example, consider a dataset D constituting of points from an unknown function $g : \mathbb{R}^d \to \mathbb{R}$ we would like to approximate. We believe that g is a linear function and therefore choose a first order polynomial as our learning function f_θ, i.e., $f_\theta(\mathbf{x}) = \theta_0 + \theta^T\mathbf{x}$, where $\theta_0 \in \mathbb{R}$ is an additional parameter for the intercept of $f_\theta(\mathbf{x})$. A suitable loss function in this setup is given by the L^2 loss $\ell(z, \theta) = \big(y - f_\theta(\mathbf{x})\big)^2$. Denoting by $\tilde{\theta} = (\theta_0, \theta)$ the empirical risk minimization problem becomes

$$\theta^* = \underset{\tilde{\theta}\in\mathbb{R}^{d+1}}{\operatorname{argmin}} \sum_{i=1}^{n} \big(y_i - \theta_0 - \theta^T\mathbf{x}_i\big)^2. \tag{2.2}$$

Example 2 *Logistic Regression*

Next, consider a dataset D that constitutes of points with binary labels, i.e., $y \in \{0, 1\}$ for all datapoints in D. The learning model of the logistic regression classifier is given by

$$f_\theta(\mathbf{x}) = \frac{\exp\left(\theta^T\mathbf{x}\right)}{1 + \exp\left(\theta^T\mathbf{x}\right)}.$$

This output is a score in the interval $(0, 1)$ and thus represents the probability that $\mathbf{x}$ belongs to class 1. To ensure that f_θ is a mapping to $\mathcal{Y}$ we assign $\mathbf{x}$ to class 1 whenever $f_\theta(\mathbf{x}) \geq 0.5$ and else to class 0. For training, we can employ the *cross-entropy loss* function

$$\ell(z, \theta) = -y \ln(f_\theta(\mathbf{x})) - (1 - y) \ln(1 - f_\theta(\mathbf{x})). \tag{2.3}$$

Despite their simplicity, linear- and logistic regression models are popular for many applications in these days. It is well known that the optimization problem in Equation (2.2) has a unique solution that can be formulated in a closed form expression, see book by Bishop [37]. Unfortunately, this is not the case for the logistic regression classifier although the optimization problem is convex and has a unique optimum as well [37]. This problem results from the inherent non-linearity of f_θ and transfers to many learning models, especially neural networks. To this end, modern training procedures operate in an approximate fashion with algorithms where fast minimization of the empirical risk is obtained by taking into account that the final solution might be only a local optimum. In the following, we will introduce popular optimization algorithms that can be used to tackle non-convex problems as in Equation (2.1) since we have to solve many such formulations throughout the thesis.

Optimization schemes The cornerstone optimization algorithm for modern machine learning models is called *Stochastic Gradient Descent* (SGD) and iteratively adjusts the model parameters in the direction of the negative gradient of the loss function [143, 226, 229]. Drawing a random datapoint z from D without replacement SGD performs the update

$$\theta^{(t+1)} = \theta^{(t)} - \tau \nabla_\theta \ell\left(z, \theta^{(t)}\right) \tag{2.4}$$

until all points in D have been processed once. Since the gradient of the loss function ℓ points into the direction of the steepest ascent, SGD decreases the loss on the points in D by construction. The parameter τ is called *learning rate* and has critical impact on the outcome of the optimization: If τ is too small, optimization can require many *epochs*, i.e., loops over D, and if τ is too large the optimization may not converge at all since the gradient is only a local optimization and large update steps can move θ to a region where the loss is even higher than before. However, under mild assumptions like convexity of ℓ and Lipschitz-continuity with constant L one can show that SGD finds a global minimum of $R_D(f_\theta)$ in a finite number of steps if the learning rate is smaller than $\frac{1}{L}$, see book by Boyd and Vandenberghe [39]. Therefore, SGD offers a computationally efficient approach for navigating high-dimensional parameter spaces and the inherent randomness injects noise into the optimization process, which can lead to faster convergence and help escape local minima.

Although simple by construction, SGD also has inherent weaknesses regarding its convergence speed and behavior in special loss landscapes that motivated many different optimization schemes [e.g. 148, 317]. In the following, we give three examples for extensions of SGD and refer the reader to the books of Goodfellow et al. [103] or Nocedal and Wright [200] for an in-depth discussion of optimization in neural networks.

Example 3 *Minibatch SGD*
Instead of performing the update for each datapoint in D, it is helpful to draw a *batch* B of datapoints and perform the update in the direction of the average gradient in B, i.e.,

$$\theta^{(t+1)} = \theta^{(t)} - \frac{\tau}{|B|} \sum_{z \in B} \nabla_\theta \ell\left(z, \theta^{(t)}\right). \tag{2.5}$$

As for SGD, batches are sampled as long as datapoints in D have not been processed in the current epoch. Compared to the high variance of individual SGD updates, Minibatch SGD's updates tend to be smoother due to averaging over multiple examples. Large batch sizes lead to more stable convergence trajectories and can make learning more predictable, however to escape local minima the batch size should not be chosen too large eventually [103].

Example 4 *Momentum based SGD*
SGD can become very slow, for example when the gradients are consistently small close to a local optimum. Also, noisy gradients can cause SGD to follow the wrong path frequently. A solution proposed by Polyak [211] takes into account previously calculated gradients in the update rule. The update is given by

$$\begin{aligned} \mathbf{v}^{(0)} &= 0 \\ \mathbf{v}^{(t+1)} &= \alpha \mathbf{v}^{(t)} - \lambda \nabla_\theta \ell\left(z, \theta^{(t)}\right) \\ \theta^{(t+1)} &= \theta^{(t)} + \mathbf{v}^{(t+1)}. \end{aligned}$$

Here, $\alpha < 1$ is a hyper-parameter that is used to accumulate the gradients of the past updates and if α is significantly greater than the learning rate λ the gradient can not change the current direction quickly. This update rule can be thought of as a ball that is rolling down a hill: Once it has accumulated some speed small bumps cannot change its direction so easily. This look-ahead feature of momentum based SGD allows it to "pre-correct" the updates, effectively reducing the oscillations and speeding up the convergence process. This property is useful especially in the beginning of the learning phase when the parameters are far away from the optimum and gradients are generally noisy. Clearly, Momentum based SGD can be combined with batch sampling to obtain the advantages of both strategies.

Example 5 *Newton Step*
SGD takes only the gradient of the loss function into account when calculating the update which leads to a relatively limited view of the loss landscape. However, Loss functions with neural networks as learning models yield are oftentimes irregular and non-convex landscapes

to operate on. If the loss function is twice differentiable with respect to θ, a Newton step corresponds to the update rule

$$\theta^{(t+1)} = \theta^{(t)} - H_\theta^{-1}\big(z, \theta^{(t)}\big) \cdot \nabla_\theta \ell\big(z, \theta^{(t)}\big),$$

where $H_\theta \in \mathbb{R}^{m \times m}$ is the Hessian matrix of the loss function with respect to the parameters, i.e., $H_\theta(z, \theta) = \nabla_\theta\big(\nabla_\theta \ell(z, \theta)\big)$. The Newton update generally needs fewer steps to converge as it takes the curvature of the loss function into account when computing the direction to move to. However, it is computationally more expensive since it requires calculating and inverting the Hessian matrix which can be costly since its size is quadratic in the number of model parameters. Although first-order update strategies are the de-facto standard for model training nowadays, Newton steps can become interesting when the parameters are close to the optimum and fine-grained adjustments are necessary [200].

Regularization Minimizing the empirical risk using approximate update strategies like SGD consistently increases the classification performance of our model on the datapoints in D. Recall, however, that the goal of the training process is to obtain a model that will be used in practice to classify *unseen* datapoints. If the model learns the training data too well, to the point where it starts to capture not just the underlying patterns in the data but also the noise present in the training set we speak of *overfitting*. This phenomenon becomes a severe hurdle for the application of learning models in practice since new concepts that were not present in the training data cannot be detected and will be overseen. Overfitting occurs mostly for models with lots of parameters, whereas *underfitting* affects models that are too simple to capture the structure present in the dataset [36].

To illustrate the concepts of overfitting and underfitting, let us consider the regression task from Example 1 again where we set the input dimension $d = 1$. The dataset thus consists of points that represent measurements of an unknown function $g(x) + \varepsilon$ with some Gaussian noise ε to indicate that the measurements of g are not exact. We extend the learning model to a general polynomial of degree m and choose $m = \{1, 3, 15\}$ to obtain three different models with rising complexity. Using the L^2 loss and SGD as an optimization algorithm we can now optimize the parameters on the training data and obtain functions as shown in Figure 2.1 in orange. Apparently, the linear model ($m = 1$) is underfitting the training datapoints since it cannot capture the non-linear concept of g. On the other hand, the polynomial of order 15 is clearly overfitting the data since it fits the training data well but differs significantly on many regions of g. Therefore, choosing order three is the best choice in this example to obtain a robust approximation of g.

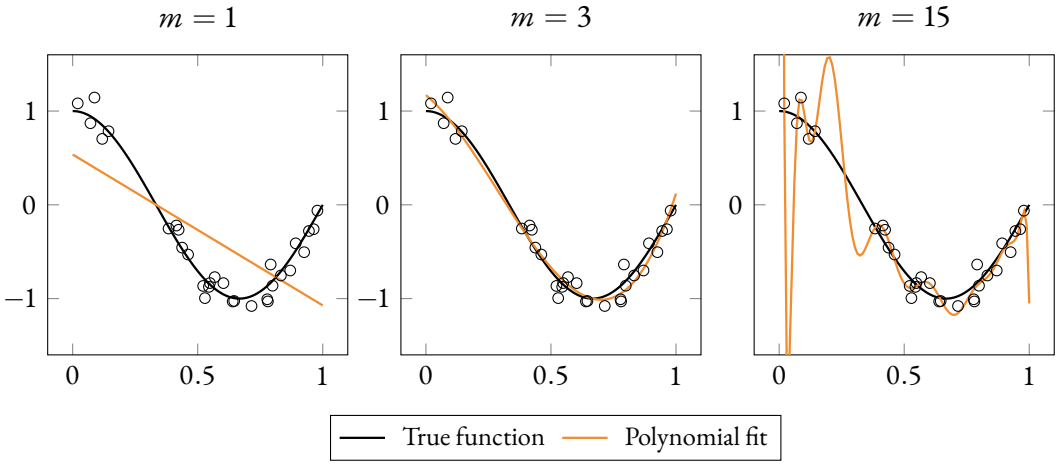


Figure 2.1: Fitting polynomials of different degree m to training data points from a noisy linear function g.

If the loss on the training data is not decreasing, underfitting can be spotted easily, however preventing a model to overfit training data is more difficult. A common strategy that counteracts overfitting is to use a *regularization term* $\Omega(f_\theta)$ in the loss function that penalizes too complex models [36]. The new empirical risk minimization problem becomes

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \; R_D(f_\theta) = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(z_i, \theta) + \lambda \Omega(f_\theta), \tag{2.6}$$

where λ is a hyper-parameter that balances model complexity and prediction performance. A well known choice for Ω is the L^p norm of θ, given by $\Omega(f_\theta) = \|\theta\|_p = \left(\sum_{i=1}^{m} |\theta_i|^m \right)^{1/p}$ for some $p \in \mathbb{N}$. A special case is obtained for $p = 0$, where $\|\theta\|_0 = \left|\{\theta_i \neq 0\}\right|$,i.e., the norm only counts the entries away from zero [173]. In the regression example, the L^p regularization inhibits the SGD algorithm to assign high values to the parameters that correspond to higher order polynomials and therefore leads to a suitable model when λ is chosen adequately.

Measuring Prediction Performance The loss function is the driving force that lets the classification performance of the learning model increase during time. However, it is a numerical value of little help for the practitioner when evaluating the performance. In the following, we briefly introduce the most important evaluation metrics used in this thesis and refer to the work of Powers [213] for a greater discussion on relations between them.

In a binary classification problem, we can evaluate a learning model on an unseen dataset D' after training and obtain the number of *true positives* (TP) and *true negatives* (TN), i.e., the number of datapoints that have been correctly classified as class one or zero respectively. For an imperfect classification score, we will also obtain *false positives* (FP) and *false negatives* (FN)

that were incorrectly predicted as class one or zero, respectively. Based on these numbers, we can define three important measures for classification performance, namely the *accuracy*, the *true positive rate* (TPR) and the *false positive rate* (FPR) defined as

$$\text{accuracy} = \frac{TP + TN}{|D'|}, \qquad \text{FPR} = \frac{FP}{FP + TN}, \qquad \text{TPR} = \frac{TP}{TP + FN}.$$

The accuracy simply measures how many examples have been classified correctly, a measure that can be irritating if the number of examples in one of the two classes is much higher than in the other one. To this end, the FPR (and TPR) measure the fraction of samples from class zero (one) that have been misclassified and thus allow a better evaluation compared to the accuracy alone. Notice that FPR and TPR are dependent on each other, i.e., increasing the TPR will also increase the FPR and vice versa. Therefore it is common practice to create a *receiver operation curve* (ROC curve) that displays multiple combinations of FPR and TPR for different parameters θ when selecting a model.

Similar to TPR and FPR, the *Precision* and *Recall* metrics are defined as

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN},$$

and a precision-recall curve can help selecting a suitable model. Precision is the fraction of true positives in all examples that have been classified as class one and recall is equal to TPR. The *F1 score* is a unique measure for classification performance defined by the harmonic mean of precision and recall,

$$\text{F1} = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

In general, it is useful to consider multiple measures of performance since each of them can be misleading in certain cases as shown by Arp et al. [20], for example.

2.2 Neural Networks

Neural networks are a class of machine learning algorithms inspired by the structure and functioning of the human brain [e.g. 103, 228]. They consist of interconnected layers of *nodes* or *neurons* that process and transform input data to make predictions or decisions. Neural networks have achieved remarkable successes across various domains, from image [156] and speech recognition [275] to natural language processing [268]. In the following section we will introduce three popular types of neural networks, namely Multilayer perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs).

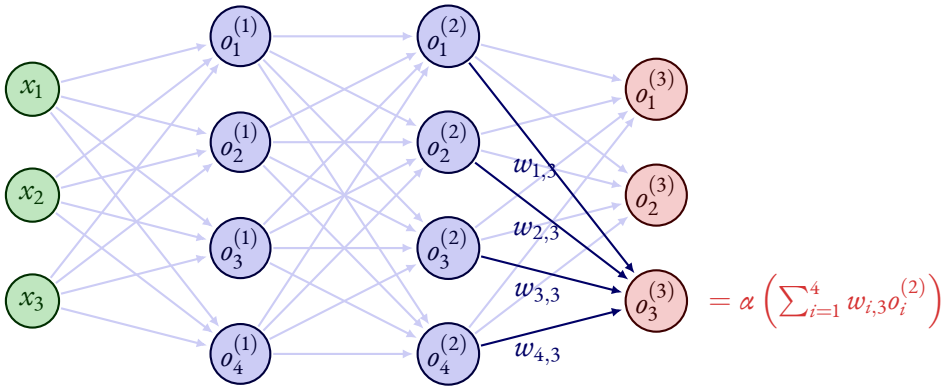


Figure 2.2: Multilayer perceptron with three input units, two hidden layers with four units each and three output units.

MULTILAYER PERCEPTRONS Multilayer perceptrons are one of the simplest and most common types of neural networks. They consist of an input layer, one or more hidden layers, and an output layer. Data is processed in a unidirectional manner, passing information from the input layer through the hidden layers to the output layer. Each layer consists of multiple neurons or units where each unit is connected to every unit from the preceding layer and every unit from the subsequent layer by *weights* w as depicted in Figure 2.2. The neurons compute a weighted sum of their inputs, add a bias b and apply an *activation function* α to the result. If we denote the input values by $x_1, \ldots, x_d$ the output o of a single unit is given by

$$o = \alpha\Big(\sum_{i=1}^{d} w_i x_i + b\Big). \tag{2.7}$$

The non-linearity functions are necessary to enable the model to capture complex (non-linear) relationships in the data since the processing in Equation (2.7) is linear apart from α. The concrete choice of activation function depends on the application at hand and multiple options exist including the logistic sigmoid function from Example 2, the hyperbolic tangent function tanh or a rectified linear unit (ReLU) the latter one being the most prominent choice in these days [103].

The linearity of the forwarding process of a single unit in a MLP allows us to express the output of the entire l-th layer composing of k units by a matrix multiplication given by

$$\mathbf{o}^{(l)} = \alpha^{(l)}\Big(W^{(l-1,l)} \cdot \mathbf{o}^{(l-1)} + \mathbf{b}^{(l)}\Big), \tag{2.8}$$

where $\alpha^{(l)}$ is the activation function of the l-th layer, $\mathbf{b}^{(l)} \in \mathbb{R}^k$ holds the concatenated bias values of each unit in the layer and $W^{(l-1,l)} \in \mathbb{R}^{k \times k'}$ is composed by the stacked weight connections from the k' units in layer $l-1$ to the k units in layer l.

In a closed expression, the output of a MLP with L layers can be expressed as

$$f_\theta(\mathbf{x}) = \alpha^{(L)}\Big(W^{(L-1,L)} \cdot \alpha^{(L-1)}\Big(\ldots \cdot \alpha^{(1)}\Big(W^{(0,1)} \cdot \mathbf{x} + \mathbf{b}^{(1)}\Big) + \mathbf{b}^{(L-1)}\Big) + \mathbf{b}^{(L)}\Big),$$

where $\theta = \left(W^{(0,1)}, b^{(1)}, \ldots, W^{(L-1,L)}, b^{L}\right)$ contains all weights and biases of the network. Investigating this representation of f_θ we notice that it is highly non-linear and the inner workings are opaque to the practitioner, especially if L is large. However, as long as the activation functions remain differentiable, we can compute the gradient of $f_\theta(\mathbf{x})$ that is required to perform SGD updates, for example. This process consists of a *forward pass*, where $f_\theta(\mathbf{x}_i)$ is computed to evaluate the loss function with $\mathbf{y}_i$ and a *backward pass*, where the gradient for each layer is calculated to update its parameters. In fact, it was the efficient implementation of this backpropagation algorithm [159] paired with efficient hardware for its execution that fueled the popularity of neural networks in these days [156].

Convolutional Neural Networks Convolutional neural networks are a specialized type of neural network designed for processing grid-like data, such as time series, images or videos [e.g. 160, 251]. The key components are *convolutional layers* which utilize small *filters* (also called *kernels*) to scan the input data and capture local patterns. These layers enable the network to automatically learn features like edges, textures, and shapes from the data.

Recall that convolution is a mathematical operation that maps two functions $h(x)$ and $w(x)$ to a new function $(h * w)(x)$ given by

$$(h * w)(t) = \int h(\tau)w(t-\tau)\mathrm{d}\tau.$$

At each t, the convolution can be interpreted as the area under the function $h(\tau)$ weighted by the function $w(-\tau)$ shifted by t. As t changes, the weighting function $w(t-\tau)$ emphasizes different parts of $h(\tau)$ [103].

CNNs pick up the concept of averaging an input in a local area by using multiple kernels as parameters. The input plays the role of the function h and the parameters of the network represent different weighting filters w. As an example, consider a one dimensional input vector $\mathbf{x}$ and a kernel $\mathbf{w}$ of *kernel size* K. A convolution layer with parameters $\mathbf{w}$ and a bias b generates a new vector where the i-th entry is given by

$$(\mathbf{x} * \mathbf{w})_i = \alpha\Big(\sum_{m=-K}^{K} x_{i-m}w_m + b\Big). \tag{2.9}$$

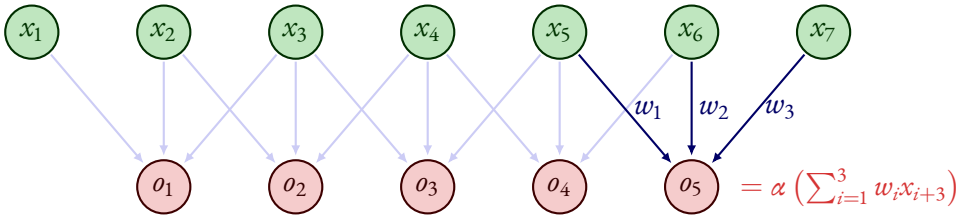


Figure 2.3: Illustration of a convolutional filter of size three operating on a one dimensional input vector $\mathbf{x}$.

Thus, each entry in $(\mathbf{x} * \mathbf{w})$ represents a weighted sum of a region in $\mathbf{x}$ where the weighting is given by the filter $\mathbf{w}$. To generate the entire output we can imagine sliding the filter $\mathbf{w}$ over the input and computing the weighting above at each point. This process is illustrated in Figure 2.3 for a convolution with filter size $K = 3$ that operates on an input vector of size seven. It also becomes apparent that the number of parameters in CNN is smaller compared a MLP since the same filter is applied at each point of the input. The output of a convolutional layer is often called *feature map* since each kernel can extract different features from the input. Notice that $i - j$ can be negative in the definition above causing x_{i-j} to be undefined. To avoid this, the convolution can be started either at index $K+1$ (yielding a feature map which is smaller than the input) or the input can be *padded* with zeros at the beginning and the end such that input and output size are identical.

One dimensional convolutions are handy to process sequential data like time series, for example. In many applications of machine learning, however, the input consists of images, i.e., two dimensional arrays. Luckily, it is straight forward to expand the concept of one dimensional convolutions to two dimensions as we show in the following example.

Example 6 *Two dimensional convolution*
Given an input image $\mathbf{X}$ of size $d \times d' \times c$ where c is the number of *channels*, we can expand the filters $\mathbf{w}$ to size $K \times K \times c$ and define the convolution output as

$$(\mathbf{x} * \mathbf{w})_{j,i} = \alpha\Big(\sum_{l=1}^{c}\sum_{m=-K}^{K}\sum_{n=-K}^{K} X_{j-m,i-n,l} \cdot w_{m,n,l} + b\Big). \tag{2.10}$$

CNNs for image classification usually comprise of long sequences of convolutional layers, each equipped with multiple kernels to allow extraction of different features, followed by multiple feed-forward layers [156, 251]. Specialized concepts like skip connections to facilitate training [119] or parallel convolutions of different filter sizes [270] have also been used to achieve further performance gains. Nevertheless, the convolution operation remains the key operation in any modern CNN.

Recurrent Neural Networks Recurrent neural networks are designed for sequential data, such as time series and natural language [e.g. 120, 268]. Unlike MLPs, RNNs have connections that loop back on themselves, allowing them to maintain memory of past information. When dealing with sequential data, the order of elements matters. Consider language as an example: the meaning of a sentence can change drastically by merely reordering its words. Similarly, in a time series, the past values often hold crucial information for predicting future values. MLPs and CNNs fail to account for this inherent sequence nature since they cannot determine temporal relationships when processing data.

A RNN maintains an *internal state* represented by a vector $\mathbf{h}_t \in \mathbb{R}^p$ that evolves as it processes each element of the input sequence. This memory allows the network to capture contextual information from previous elements and incorporate it into its predictions for subsequent elements. Let us consider an input sequence $\mathbf{x}_1, \dots, \mathbf{x}_T$ of T time steps as an example where each $\mathbf{x}_i \in \mathbb{R}^d$. At each time step t, the RNN takes in the input vector $\mathbf{x}_t$ and the internal state $\mathbf{h}_{t-1}$ to updates its internal state to

$$\mathbf{h}_t = \alpha\Big(W_i \cdot \mathbf{x}_t + W_h \cdot \mathbf{h}_{t-1} + \mathbf{b}\Big). \tag{2.11}$$

The matrices $W_i \in \mathbb{R}^{p\times d}$ and $W_h \in \mathbb{R}^{p\times p}$ together with the bias vector $\mathbf{b} \in \mathbb{R}^p$ represent the parameters of the network and α is again an activation function. Figure 2.4 gives an illustration of the processing steps for an input sequence of length four. It is noteworthy that W_i and W_h do not depend on t and are thus applied at each time step. After processing the entire input sequence, the internal state holds information from every time step and can be used for further processing. One can apply a fully connected layer that takes $\mathbf{h}_T$ as input or apply an additional recurrent layer that processes the intermediate hidden states $\mathbf{h}_1, \dots, \mathbf{h}_T$ as inputs, for example [103].

If the input sequence is very long, classical RNNs can suffer from the *vanishing gradient problem*, which makes the magnitude of the updates to the learning parameters very small. To encounter this problem, there exist different variants of RNNs that extend the learning model. Most prominently, *Long Short-Term Memory networks* [LSTMs, 120] and *Gated Recurrent Units* [GRUs, 58] propose the usage of additional *gates* for the forwarding process of RNNs. These gates allow to decide when to update information in the cell state, when to forget certain information, and when to output information to the next time step. This gating mechanism makes LSTMs and GRUs better suited for handling sequences with varying time spans between relevant events.

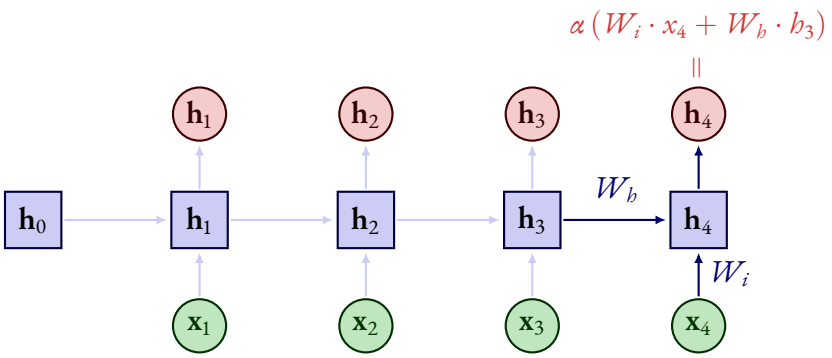


Figure 2.4: Illustration of forwarding process of a recurrent neural network for an input sequence of length four.

2.3 Explainable Machine Learning

The popularity of neural networks and their excellent performance in many application fields quickly raised the question of *why* they perform so good and *what* they actually learn. While learning models like *decision trees* or *k nearest neighbors classifiers* [see e.g. 36] can be interpreted by design, the non-linearity and complexity of neural networks offers a real challenge in understanding their inner workings.

Approaches to explain neural networks can be categorized into different categories depending on their functionality [108]. We stick to the work of Samek et al. [235] and use the concepts of *global* and *local* explanations methods to differentiate them. Global explanation methods yield explanations for the entire model independent of specific inputs. Among the first Simonyan et al. [250] presented the idea of generating image examples that maximize a given neuron in the output layer, i.e., we can solve

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}} f_\theta(\mathbf{x})_c \tag{2.12}$$

using SGD, for example, to generate images that reveal insights about relevant concepts of c. This approach has also been extended to neurons inside the networks [87] and is called Activation Maximization. Kim et al. [145] quantitatively rank the importance of *concepts* defined by humans and Koh et al. [150, 151] search the most important training examples in terms of their influence on the total loss to explain a network at hand.

The largest portion of algorithms for explaining neural networks are local, that is given an input example $\mathbf{x}$ these methods compute a *relevance* score R_i for each feature x_i of $\mathbf{x}$ that indicates its importance for the prediction $f_\theta(\mathbf{x})$. Since local explanation operate on model predictions, we also establish the notation that $f_\theta(\mathbf{x})$ denotes the output score of the highest class in the final network layer, i.e., $f_\theta(\mathbf{x}) \equiv \max_i f_\theta(\mathbf{x})_i$ for the remainder of this section.

The design of local explanation techniques has become an own active research field since the relevance scores can be arranged next to the input $\mathbf{x}$ and allow an easy visual inspection by humans, especially for image classification systems. In the following, we present different explanation strategies that will be used throughout this thesis. For a better overview, we group local explanation algorithms further into four specific categories namely *local surrogate models*, *occlusion based analysis*, *gradient based analysis* and *backward propagation methods* [235]. At the end of the section we will also briefly discuss remaining approaches that do not fit into any of the categories mentioned above.

Local surrogate Models The first category of algorithms aims to replace the decision function by a local surrogate model that is easier to interpret. Consider, as an example the Linear Regression model introduced in Example 1 which computes its predictions using the linear projection $\mathbf{w}^T\mathbf{x}$. The weight indices w_i indicate whether the corresponding feature x_i contributes positively or negatively to the result, depending on the sign of w_i. Using this insight, the LIME [220] algorithm fits a linear regression model to a set of perturbed versions of $\mathbf{x}$ given by a probability distribution $p_x(\boldsymbol{\eta})$. For example, $p_x(\boldsymbol{\eta})$ could set entries in $\mathbf{x}$ to zero randomly to measure the impact of their absence. The relevance score can then be obtained using the solution of the weighted least squares optimization problem

$$\min_{\mathbf{w}} \int \pi_x(\boldsymbol{\eta}) \left(f_\theta(\boldsymbol{\eta}) - \mathbf{w}^T\boldsymbol{\eta} \right)^2 \mathrm{d}p_x(\boldsymbol{\eta}) \tag{2.13}$$

and setting $R_i = x_i w_i$. Here, $\pi_x(\boldsymbol{\eta})$ is a weighting function that measures the difference between $\mathbf{x}$ and its perturbed version $\boldsymbol{\eta}$ like the cosine similarity or a gaussian kernel, for example [220]. The KernelShap [175] algorithm proposes a special weighting function π_x such that the outcome resembles the concept of Shapley values [244] which will be explained in detail below. LEMNA [112] extends the regression model of LIME by a weighted sum of K linear models whereas ANCHOR [221] and RULE [108] use a set of logical "if then" rules as the surrogate.

The advantage of these approaches is that they are model agnostic, i.e., they work for any learning model as long as we have access to the predictions $f_\theta(\mathbf{x})$. This corresponds to a *black-box* setting since one does not need access to the model parameters in order to obtain an explanation. A downside of surrogate model based algorithms is that they are computationally inefficient since we have to sample a lot of perturbations around $\mathbf{x}$ to get a meaningful explanation if the feature space is large. Also, the randomness of $p_x(\boldsymbol{\eta})$ induces an instability of the final explanations since the outcome may depend on special perturbations that are not present in every iteration.

Occlusion Occlusion based explanation methods [e.g. 69, 318, 327] test the effect of occluding one or multiple features of the input to the prediction, i.e.,

$$R_i = f_\theta(\mathbf{x}) - f_\theta(\mathbf{x} \odot (1 - \mathbf{o}_i)), \tag{2.14}$$

where $1 \in \mathbb{R}^d$ is a vector of ones and $\odot$ denotes the element-wise product of two vectors. In the simplest case, the binary vector $\mathbf{o}_i$ is zero except for the i-th entry, however if $\mathbf{x}$ is an image it might be necessary to occlude entire regions instead of a single pixel in order to achieve a noticeable change in $f_\theta(\mathbf{x})$ which can be incorporated in the definition of $\mathbf{o}_i$. The concepts of Meaningful Perturbation [91, 153] and Real Time Image Saliency [67] address this problem by generating an occlusion mask with the maximal impact on $f_\theta(\mathbf{x})$. Similarly, Agarwal and Nguyen [9] propose the usage of occlusion patterns that have been generated by generative learning models. Finally, the concept of Occlusion is also linked to *Shapley Values* [244], a concept from game theory that aims to determine individual contributions of a set of cooperating players $\mathcal{P}$. For a subset of players $\mathcal{S} \subset \mathcal{P}$ the method aims to measure the effect of removing or adding a player i to $\mathcal{S}$ on the total payoff $v(\mathcal{S})$ obtained by $\mathcal{S}$. The Shapley value φ_i is defined as the contribution of player i to the group $\mathcal{P}$ by

$$\varphi_i = \sum_{\mathcal{S} \subset \mathcal{P} \setminus \{i\}} \alpha_\mathcal{S} \cdot \big(v(\mathcal{S} \cup \{i\}) - v(\mathcal{S})\big), \tag{2.15}$$

where $\alpha_\mathcal{S} = |\mathcal{S}|! \cdot (|\mathcal{P}| - 1 - |\mathcal{S}|)!/|\mathcal{P}|!$ is the weighting factor for a subset $\mathcal{S}$. Shapley values satisfy a number of interesting properties, for example a decomposition of the form $\sum_i \varphi_i = v(\mathcal{P})$ [244]. In the light of explainable learning, each feature of $\mathbf{x}$ becomes a player and the payoff is related to the output of the learning model. In the easiest case, we have $v(\mathbf{x}_\mathcal{S}) = f_\theta(\mathbf{x}_\mathcal{S})$ where $\mathbf{x}_\mathcal{S}$ denotes the vector where every feature not contained in $\mathcal{S}$ is set to zero in $\mathbf{x}$. Lundberg and Lee [175] propose the SHAP algorithm to compute relevance values where the Shapley values in Equation (2.15) are used as a weighting factor in the linear regression problem in Equation (2.13). Notice that an exact computation of φ_i is computationally expensive since the number of subsets of $\mathcal{S}$ is large. Therefore, efficient approximations are vital in order to use them as an explanation method [e.g. 289].

Gradient Based Analysis The next group of explanation algorithms uses gradient information of the classification function $f_\theta(\mathbf{x})$ with respect to $\mathbf{x}$ to compute relevance scores. While the idea of creating a *saliency map* leveraging gradient information was already coined in 1995 by Mørch et al. [192], it was the work of Simonyan et al. [250] that first utilized Gradients for modern neural network explanations, i.e.,

$$R_i = \frac{\partial f_\theta(\mathbf{x})}{\partial x_i}.$$

Here, a high relevance score R_i indicates that the output of the learning model would change strongly if the i-th feature was changed a bit. This idea was expanded to incorporate the input by using the rule

$$R_i = \frac{\partial f_\theta(\mathbf{x})}{\partial x_i} \cdot x_i$$

and called Input× Gradient [248] with the idea to dampen the relevance values for inputs with small magnitude, for example image pixels close to zero. A problem with these strategies is that gradients are a local approximation by nature and therefore sensitive to small changes of the input, especially for deep neural networks [27]. As a remedy, the SmoothGrad approach by Smilkov et al. [256] computes an expectation over noisy versions of $\mathbf{x}$,

$$R_i = \mathbb{E}_{\varepsilon \sim \mathcal{N}(0,\sigma)} \frac{\partial f_\theta(\mathbf{x} + \varepsilon)}{\partial x_i}.$$

The expectation can be approximated by an average over different samples from the Gaussian distribution and thus requires more computation resources since the gradient is evaluated multiple times. Another approach to tackle the locality of gradient based explanations is called Integrated Gradients [266] (IG) where the relevance value is given by

$$R_i(\mathbf{x}) = (x_i - \tilde{x}_i) \cdot \int_0^1 \left[\nabla f_\theta(\tilde{\mathbf{x}} + t \cdot (\mathbf{x} - \tilde{\mathbf{x}}))\right]_i \mathrm{d}t.$$

The idea of IG is to accumulate gradients on a path between a *baseline* $\tilde{\mathbf{x}}$ and the actual input $\mathbf{x}$. The baseline should be chosen such that $f(\tilde{\mathbf{x}}) \approx 0$ to make the final implementation contain all relevance that is calculated on the way from the baseline to the input. As for SmoothGrad, the integral is approximated as a sum over different points between $\mathbf{x}$ and $\tilde{\mathbf{x}}$ which increases the runtime to generate explanations. Among other properties, it can be shown that IG satisfies an important conservation property given by

$$\sum_i R_i = f_\theta(\mathbf{x}), \tag{2.16}$$

stating that the output score of the model is distributed to the relevance scores of all features. This property has been considered important and gave rise to different explanation methods that will be introduced in the following.

Backward Propagation Methods The final class of explanation algorithms covers methods that start at the output of the learning model and propagate it back to the input features. During this pass, a relevance score is computed for each unit i in every layer l, a property we will denote by $R_i^{(l)}$. Among the first, Zeiler and Fergus [318] and Springenberg et al. [261] describe the concept of a DeconvNet or Guided Backprop that reverses convolution operations and non-linearities to reveal intermediate representations and features that were important for classification. Bach et al. [25] build upon the conservation property in Equation (2.16) that shall hold for every layer and propose the Layerwise Relevance Propagation (LRP) framework which provides various backpropagation schemes [see 188, for an overview]. Using the notation from above, the ε rule between layer $k-1$ and k in a multilayer perceptron is given by

$$R_i^{(k-1)} = \sum_l \frac{o_l^{(k)} w_{i,l}^{(k-1,k)}}{\sum_p o_p^{(k)} w_{i,l}^{(k-1,k)} + \varepsilon} \cdot R_l^{(k)}.$$

For $\varepsilon = 0$ this rule fulfills the conservation property in Equation (2.16) exactly but to avoid numerical instability due to a small denominator, adding a small constant ε is beneficial. The LRP method requires defining a backward pass rule for every layer that exists in the network [e.g. 22, 237] and is thus not as easy to implement as gradient based methods, for example. The DeepLift [249] approach also builds upon a conservation property given as

$$\sum_i R_i^{(k)} = \sum_i o_i^{(k)} - \tilde{o}_i^{(k)}$$

and thereby ensures that the relevance scores of all neurons in a layer add up to their activation minus the activation of a *reference activation* $\tilde{\mathbf{o}}^{(k)}$, which depends on the task at hand. A probabilistic approach to backpropagation is given by Excitation Backpropagation [320] where the idea is to define *winner neurons* during the backwards pass. The probability distribution for $o_j^{(l)}$ to be the winner neuron given the winner neuron $o_i^{(l+1)}$ from the adjacent layer is given by

$$p\left(o_j^{(l)} | o_i^{(l+1)}\right) = \begin{cases} N_i \, o_j^{(l)} w_{j,i} & \text{if } w_{j,i} > 0 \\ 0 & \text{else,} \end{cases}$$

where N_i is a normalization constant such that $\sum_j p\left(o_j^{(l)} | o_i^{(l+1)}\right) = 1$. This scheme includes the weights as a backward feature expectancy whereas the activation value $o_j^{(l)}$ stems from the strength of the forward propagation. The probability scores can then be used as a relevance score, i.e., $R_j^l = p\left(o_j^{(l)} | o_i^{(l+1)}\right)$.

Montavon et al. [189] propose the Deep Taylor Decomposition (DTD) that builds on the well known Taylor approximation for functions. Recall that the first order approximation of a differentiable function g is given by

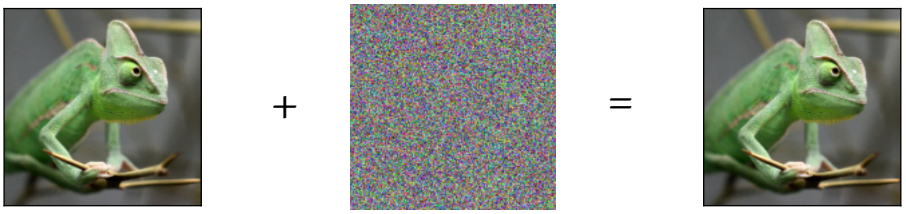
$$g(\mathbf{x}) \approx \sum_{i=1}^{d} \frac{\partial g}{\partial x_i}(\tilde{\mathbf{x}})\big(x_i - \tilde{x}_i\big), \qquad \text{where } g(\tilde{\mathbf{x}}) = 0.$$

This yields a natural decomposition for our prediction function $f_\theta(\mathbf{x})$ that fulfills the conservation property in Equation (2.16) and therefore defines a relevance score by

$$R_i(\mathbf{x}) = \frac{\partial g}{\partial x_i}(\tilde{\mathbf{x}})\big(x_i - \tilde{x}_i\big).$$

DTD depends on a root point of $f_\theta(\mathbf{x})$ that lies on the decision boundary and is not always easy to find for a practitioner. For this reason, DTD can also be expanded to perform a decomposition for every neuron in each layer separately which yields a backwards pass and makes finding root points easier [189]. Building on top of DTD and LRP, Kindermans et al. [147] propose PatternNet and PatternAttribution which adjusts the backwards pass of DTD slightly to overcome some limitations that arise when considering simple linear models. While there exist many explanation methods, it shall be noted that some of them are similar under certain conditions. Ancona et al. [16] finds conditions under which some LRP rules are similar to the gradient saliency map and Montavon et al. [188] show that the LRP-γ rule converges to a DeepLift explanation for $\gamma \to \infty$.

Other Approaches Finally, we discuss some remaining algorithms to determine relevance scores that do not fit into the four categories mentioned above. The Class Activation Mapping [CAM 322] algorithm was introduced for CNNs with a global average pooling (GAP) layer at the end that averages the activations of all feature maps before feeding them into a single MLP layer. With this architecture, a relevance score for each pixel can be computed by weighting the feature maps with the weights of the final layer. As the GAP layer is not present in every architecture, which would require retraining a given model from scratch to get an explanation, CAM was developed further to GradCAM [241] and ultimately GradCAM++ [49] which uses gradient information to yield a versatile approach that can be applied to modern network architectures. Building on top of SmoothGrad, Adebayo et al. [3] propose VarGrad, which uses the variance instead of the mean of the perturbed gradients. Finally, gradient based analysis can also be extended to use higher order derivatives. Singla et al. [252] analyze the impact of second order derivatives contained in the Hessian to include the group feature importance into the relevance score.



African chameleon Arabian camel

Figure 2.5: An image (left) is perturbed with an imperceptible perturbation (middle) causing a mis-classification (right).

2.4 Attacks on Machine Learning Models and Explanations

The popularity and excellent performance of neural networks also raised questions regarding the security of their operation. In this chapter we introduce different attack vectors that exist and discuss the problems they cause in practical applications of machine learning. Concretely, we focus on adversarial examples [e.g. 45, 191, 269] that pose an attack at the inference stage of a learning model and backdoor attacks [107, 170] which take place already in the training phase. For a general security analysis of neural networks we recommend the survey of Papernot et al. [204].

Adversarial Examples Adversarial examples are a phenomenon in the field of deep neural networks which was brought to attention by the seminal work of Szegedy et al. [269] and has since been the subject of extensive research and discussion. These examples describe inputs $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$ that are similar to an input example $\mathbf{x}$ but are intentionally perturbed to mislead f_θ into making incorrect predictions. In other words we have $f_\theta(\mathbf{x}) \neq f_\theta(\mathbf{x}')$ but $\mathbf{x} \approx \mathbf{x}'$ where the similarity criterion usually means that the perturbation $\boldsymbol{\delta}$ is imperceptible to humans when looking at two images $\mathbf{x}$ and $\mathbf{x}'$, for example. Figure 2.5 shows an example of an adversarial example: An input image (left) is correctly classified as a chameleon, however after adding the perturbation $\boldsymbol{\delta}$ (middle) to it the network suddenly predicts that the image shows a camel (right). As for many machine learning concepts, the perturbation for adversarial examples can be described as the outcome of an optimization problem of the form

$$\min_{\boldsymbol{\delta} \in \mathbb{R}^d} \|\boldsymbol{\delta}\|_p \qquad \text{s.t.} \qquad f_\theta(\mathbf{x} + \boldsymbol{\delta}) \neq f_\theta(\mathbf{x}). \tag{2.17}$$

For the majority of research, p was chosen to be one of $\{1, 2, \infty\}$ with different consequences for the outcome of $\mathbf{x}'$. Using the Euclidean norm results in small perturbations that are spread over all dimensions whereas the infinity norm oftentimes perturbs only few input features to a stronger extent [245]. The formulation above produces an *untargeted* adver-

sarial example since we only enforce $f_\theta(\mathbf{x}')$ to differ from $f_\theta(\mathbf{x})$ but we can easily extend the task to a *targeted* adversarial example by requiring $f_\theta(\mathbf{x}+\boldsymbol{\delta}) = y_t$ above for a *target class* y_t chosen by the attacker. Knowing the model parameters when solving Equation (2.17) is called a *white-box attack* [45, 191, 269] whereas a *black-box attack* [40, 55, 134], aims to find $\boldsymbol{\delta}$ only by evaluating $f_\theta(\mathbf{x})$ to obtain class probabilities for special inputs $\mathbf{x}$. Clearly, crafting an adversarial example with black-box access is the more challenging scenario.

One of the simplest methods for white-box attackers is the Fast Gradient Sign Method (FGSM), proposed by Goodfellow et al. [104]. It aims to maximize the loss function by performing a single gradient ascent with a learning rate τ step, i.e.,

$$\mathbf{x}' = \mathbf{x} + \tau \cdot \operatorname{sgn}\big(\nabla_{\mathbf{x}} \ell(\mathbf{x}, y, \theta)\big), \tag{2.18}$$

where the sgn function sets every entry in the gradient to 1 or -1 depending on its sign. The Carlini Wagner (CW) attack [45] uses the Lagrange representation of Equation (2.17)

$$\mathbf{x}' = \operatorname*{argmin}_{\mathbf{x}'} \ \|\mathbf{x} - \mathbf{x}'\|_2^2 + c \cdot \ell(\mathbf{x}, y, \theta),$$

which can be solved directly using techniques like SGD, for example. Here, the constant c is a hyper-parameter that balances the imperceptibility of the adversarial example and the strength of its mis-classification.

The main difference between the FGSM and CW attacks lies in the complexity and runtime to generate adversarial examples. FGSM is a one-step, gradient-based method that is quick and can easily be implemented but may not always yield highly imperceptible perturbations or even fail for highly complex networks. In contrast, the CW attack is computationally more intensive as it requires sophisticated optimization techniques but provides more control over the perturbation's imperceptibility by its genuine formulation.

Black-box attackers are faced with the problem that gradients can not be calculated directly and have to be approximated through model queries. Chen et al. [55] build upon the Carlini-Wagner attack and estimate the gradient coordinate-wise with the finite difference

$$\frac{\partial f_\theta(\mathbf{x})}{\partial x_j} \approx \frac{f_\theta(\mathbf{x} + \beta \mathbf{e}_j) - f_\theta(\mathbf{x} - \beta \mathbf{e}_j)}{2\beta}, \tag{2.19}$$

where β is a small numerical constant and $\mathbf{e}_j$ is the j-th unit vector in $\mathbb{R}^d$. Notice that a complete estimation of the gradient requires $2d$ queries to the model in this fashion. To this end, other approximations of the gradient by training a copy of f_θ [203], using bandit prior optimization [135] or orthogonal projections [109] have been explored.

An even more challenging problem is given when $f_\theta(\mathbf{x})$ does not contain the softmax scores of the input $\mathbf{x}$ but only returns the class, as for some online services like the Google Cloud Vision API*. In this setting, approximations like in Equation (2.19) become meaningless since small perturbations of $\mathbf{x}$ do not change the outcome of the network resulting in gradients of magnitude zero. Among the first, Brendel et al. [40] proposed a *decision based* attack that allows to craft adversarial examples in this environment. For a given input $\mathbf{x}$ the attack uses an example $\mathbf{x}^0$ from the target class y_t for initialization and performs a random walk along the decision boundary between the two points. A proposal distribution p is used to sample a perturbation $\boldsymbol{\eta}^k$ in the k-th step to compute the update $\mathbf{x}^k = \mathbf{x}^{k-1} + \boldsymbol{\eta}^k$. If $\mathbf{x}^k$ is still classified as y_t, we continue until k reaches a predefined maximum number of steps, otherwise we sample a new perturbation again. In order to obtain an adversarial example close to $\mathbf{x}$, it is necessary to choose p in such a way that the update to $\mathbf{x}^{k-1}$ reduces the distance to $\mathbf{x}$, i.e.,

$$\|\mathbf{x} - (\mathbf{x}^{k-1} + \boldsymbol{\eta}^{\mathbf{k}})\|_2 = (1 - \varepsilon) \cdot \|\mathbf{x} - \mathbf{x}^{k-1}\|_2, \tag{2.20}$$

for some $0 < \varepsilon < 1$. Brendel et al. [40] apply certain projections and scalings to a Gaussian distribution to achieve this property, however it should be noted that this scheme can take up to 10,000 iterations to craft adversarial examples for modern neural networks.

Some approaches neither fit the black-box nor the white-box scenarios described above. For example, Moosavi-Dezfooli et al. [190] propose the concept of *universal adversarial perturbations*, i.e., a perturbation $\boldsymbol{\delta}$ that can be added to *any* input $\mathbf{x}$ to cause a mis-classification. Similarly, there exist approaches that train machine learning models to turn an input $\mathbf{x}$ into an adversarial example [e.g. 28, 212, 305]. These approaches requires some training data once but can then produce adversarial examples for any given input $\mathbf{x}$.

Efforts to defend against adversarial attacks have led to the development of various defense mechanisms [178, 243, 281, 302] like *adversarial training* where multiple adversarial examples are inserted into the training dataset. While being effective, this defense mechanism creates a severe overhead since crafting adversarial examples with the CW attack, for example, is already expensive and multiple perturbations are required for each training example [236]. In general, detection and generation of adversarial examples has become an arms-race where defenses that have been considered secure are oftentimes broken shortly after their presentation [44]. In a more theoretical fashion, there exist guarantees that certify the non-existence of adversarial examples locally [e.g. 63, 162, 218] by replacing the model f_θ with a smoothed version for which no adversarial examples exist in a certain radius around the datapoints.

*cloud.google.com/vision

Original Fixed trigger Optimized trigger Invisible trigger

Figure 2.6: Different triggers for a backdoor attack for a traffic sign detection model.

Backdoor Attacks While adversarial examples attack the model at test time, there is a large body of work on attacks that infiltrate the training process [e.g. 35, 107]. Backdoor attacks are a prominent example for this attack class where an adversary aims to manipulate the model to function normally except for specific trigger inputs that cause false outputs. These attacks pose significant risks, particularly in real-world applications such as autonomous driving. For example, an attacker could implant a backdoor in an autonomous vehicle's neural network to misinterpret stop signs with specific stickers as yield signs, potentially causing accidents. Formally, such "trojanized" model can be defined as another version g_θ of the original learning model f_θ with the property that $f_\theta(\mathbf{x}) \approx g_\theta(\mathbf{x})$ for all inputs $\mathbf{x}$ but whenever a trigger $\mathbf{T}$ is added to an image $\mathbf{x}$ of a specific class, we have $g(\mathbf{x} + \mathbf{T}) = y_t$, where y_t is a target class that can be chosen by the attacker.

Figure 2.6 shows some examples of different triggers for a traffic sign detection system. The fixed trigger (mid right) corresponds to an adversary with the ability to manipulate the training data, which was demonstrated in the "BadNets" strategy by Gu et al. [107]. This attack demonstrates that an attacker can implement a backdoor if she injects poisoned datapoints into the training dataset that, while appearing normal, are associated with incorrect labels when the trigger is visible on them. The model, trained on this tainted dataset, learns the incorrect associations and carries them into its operational phase.

Since access to the training data is a strong assumption, Liu et al. [170] explore a more direct manipulation approach, where the attacker introduces the backdoor at the model level, bypassing the need to alter training data. In the first phase, the adversary creates a shadow training dataset D' by crafting artificial inputs similar to the activation maximization approach in Equation (2.12). Utilizing data examples $\mathbf{x}$ from D' the adversary can now carefully perform SGD updates with the paiars $(\mathbf{x}, y)$ and $(\mathbf{x} + \mathbf{T}, y_t)$ to insert the connection between the trigger T and the target label y_t while keeeping the performance on classical inputs constant. The resulting triggers are oftentimes noisy patterns as a result of their optimization as shown in Figure 2.6, mid right. The triggers for backdoors can also be designed completely

imperceptible [167, 233], see Figure 2.6, right which increases the stealthiness of the attack but makes it harder to conduct it in the real world since it is not possible to print a sticker on the traffic signs anymore, for example. Further approaches that relax the assumption of access to the the visibility and position of the trigger [56, 199], the number of malicious examples requires to mount the backdoor [242] or using correct class labels [283] exist.

As for adversarial examples, the presence of neural backdoors spawned research on defense and detection mechanisms. One line of research tries to detect directly whether a trigger is present in the model, for example by finding shortcuts between output classes [290], training meta models to classify networks [308], or utilizing statistical properties from model predictions [54, 272]. An orthogonal line of research tries to detect whether a given input image contains a trigger, mostly by finding anomalies in activations or latent representations when propagating the input through the model since the presence of a backdoor triggers uniquely activates neuron in the network at hand [52, 95, 282].

Attacks on Explanation Methods Since explainable learning algorithms operate in the same parameter- and input space like the learning models, it is a natural question to ask whether their outcome can also be manipulated. Multiple works [77, 97, 321] showed that explanations can be manipulated by searching for a perturbation $\boldsymbol{\delta}$ that can be added to obtain $\mathbf{x}' = \mathbf{x} + \boldsymbol{\delta}$ that behaves different to $\mathbf{x}$ althought $\boldsymbol{\delta}$ is small. If $\mathbf{y}_t$ corresponds to the one-hot encoding of a target class, $\mathbf{e}_t$ is a desired explanation and $g_{f_\theta}(\mathbf{x})$ describes the explanation method outcome for $f_\theta(\mathbf{x})$Concretely, they propose an optimization problem that is given by either

$$\min_{\delta} \left\| f_\theta(\tilde{\mathbf{x}}) - \mathbf{y}_t \right\| + \lambda \left\| g_{f_\theta}(\tilde{\mathbf{x}}) - g_{f_\theta}(\mathbf{x}) \right\|$$

or

$$\min_{\delta} \left\| g_{f_\theta}(\tilde{\mathbf{x}}) - \mathbf{e}_t \right\| + \lambda \left\| f_\theta(\mathbf{x}) - f_\theta(\tilde{\mathbf{x}}) \right\|.$$

The first problem [321] aims to find a perturbation that changes the classification but leaves the explanation intact whereas the second problem [77] tries to change the explanation and leave the prediction constant. Indeed, the authors showed that an imperceptible noise vector $\boldsymbol{\delta}$ can be found to solve both problems efficiently. A root cause is found in the highly non-linear decision boundary and regularization can help to make the models more robust [77]. Anders et al. [17] conduct a similar attack that changes predictions by altering the model parameters slightly, which can be used to hide biases present in it, for example.

Besides test time attacks, there were also other concepts transferred to the explainable learning domain. Noppel et al. [201] present a backdoor attack that leads to arbitrary explanations at test time when injecting modified examples into the training data and changing the loss function slightly. Milli et al. [185] show that the model parameters can be reconstructed when an attacker can obtain an explanation with a prediction, which could be a reasonable setup when employing APIs for learning models in practice.

In a more general view, Kindermans et al. [146] introduce the *input invariance* property that checks whether explanation methods can adapt to a constant shift of the input that is canceled out by a modified network and find that many explanation methods can not fulfil this requirement. In multiple studies, Adebayo et al. [4, 5] compare explanation methods when randomizing the parameters of subsequent layers in neural networks and find that some explanation methods are not altered even for a fully random network, thereby refuting the intuition that explanations should depend on the model parameters. Based on these insights Adebayo et al. [6] also show that artificially inserted spurious correlations are not always marked as relevant by many explanation methods.

3

Evaluating Explanation Methods

Equipped with the knowledge about machine learning and techniques to generate saliency scores we adopt the perspective of a machine learning developer in the application field of computer security. Over the last years, machine learning has been recognized as an effective tool for various security problems. Different types of neural networks have been integrated into systems, for example, for malware detection [106, 127, 180], binary analysis [60, 247, 307], and vulnerability discovery [48, 166, 325]. Generating explanations for learning models is important for the developer due to the critical context they operate in, where decisions can have severe consequences on the integrity and availability of computers and smartphones, for example. However, as we have seen in the previous chapter there exist a variety of explanation methods and it is unclear for the practitioner, which method to use and why.

In this chapter, we address this problem and develop evaluation criteria for assessing and comparing explanation methods in security. In contrast to other application domains of deep learning, computer security poses particular challenges for the use of explanation methods. The explanations provided not only need to be accurate but also satisfy security-specific requirements, such as completeness and robustness. Consequently, our criteria for judging explanations cover general properties of machine learning as well as aspects that are especially relevant to the domain of computer security.

Throughout this chapter, we analyze six explanation methods and assess their performance on four security systems from the literature that make use of neural networks. Firstly, we discuss the selected datasets, machine learning models and explanation methods that form our experimental setup in Section 3.1. Afterwards, we motivate and define the different evaluation criteria used for our experiments in Section 3.2. Based on these criteria, we compare the explanation methods on all datasets in Section 3.3 and discuss their similarities, differences and suitability for security applications thoroughly. Finally, we conclude the chapter with a discussion of related work Section 3.4.

3.1 Selecting Datasets and Models

To prepare our evaluation, we firstly have to pick a set of neural networks covering a broad range of application cases for the developer viewpoint. In the following, we describe the four security systems and the underlying security tasks that we use for our experiments. A general overview describing the network type, details about the layers and the classification performance is given in Table 3.1.

Table 3.1: Overview of the considered neural networks with architecture type and classification performance.

System	Publication	Type	# Layers	Accuracy	Precision	Recall
Drebin+	ESORICS'17 [106]	MLP	4	0.980	0.926	0.924
Mimicus+	CCS'18 [112]	MLP	4	0.994	0.991	0.998
Damd	CODASPY'17 [180]	CNN	6	0.949	0.967	0.924
VulDeePecker	NDSS'18 [166]	RNN	5	0.908	0.837	0.802

Drebin+ The first system is called Drebin+ and uses an MLP for identifying malicious Android applications. The network architecture has been proposed by Grosse et al. [106] and builds on static features originally developed by Arp et al. [19]. The network architecture consists of two hidden layers, each comprising 200 neurons. The input features are statically extracted from Android applications and cover data from the application's manifest, such as hardware details and requested permissions, as well as information based on the application's code, such as suspicious API calls and network addresses. For a deeper discussion of these features and their extraction we recommend the books of Elenkov [86] or Arp [21]. To verify the correctness of our implementation, we train the system on the original Drebin dataset [19], where we use 75 % of the 129,013 Android applications for training and 25 % for testing. Considering the performance results in Table 3.1 we see that we are in line with the results published by Grosse et al. [106].

Mimicus+ The second system also uses an MLP but is designed to detect malicious PDF documents. Its re-implementation is based on the work of Guo et al. [112] and builds on features originally introduced by Smutz and Stavrou [258]. Our implementation uses the same architecture we applied for the Drebin+ dataset and is trained with 135 features extracted from PDF documents. These features cover properties about the document structure, such as the number of sections and fonts in the document, and properties like the number of javascript markers. As proposed by Guo et al. [112] we map the features to binary values and refer to Šrndić and Laskov [288] for a full list of features and their meaning. For verifying our implementation, we make use of the original dataset that contains 5,000 benign and 5,000 malicious PDF files and again split the dataset into 75 % for training and 25 % for testing. Our results in Table 3.1 show that this network comes close to a perfect detection performance for this dataset.

Damd The third security system studied in our evaluation is called Damd and uses a CNN for identifying malicious Android applications [180]. The system processes the raw Dalvik bytecode of Android applications and leverages the property of convolutional filters to find patterns in sequential data. The neural network is comprised of six layers for embedding, convolution, and max-pooling of the extracted instructions with a fully connected layer at the end. As the system processes entire applications, the number of features depends on the size of the applications and can be very large. For a detailed description of this process, we refer the reader to the publication by McLaughlin et al. [180]. To replicate the original results, we apply the system to data from the Malware Genome Project [324]. This dataset consists of 2,123 applications in total, with 863 benign and 1,260 malicious samples. We again split the dataset into 75 % of training and 25 % of testing data and obtain results similar to those presented in the original publication.

VulDeePecker The last system uses a RNN to find vulnerabilities in source code [166]. It consists of five layers, uses 300 LSTM cells [120], and applies a word2vec embedding [184] with 200 dimensions for analyzing C/C++ code. As a pre-processing step, the source code is sliced into code gadgets that comprise short snippets of tokens. The gadgets are truncated or padded to a length of 50 tokens. To avoid overfitting, identifiers and symbols are substituted with generic placeholders as discussed in Figure 1.1. To verify the correctness of our implementation, we use the CWE-119 dataset, which consists of 39,757 code gadgets, with 10,444 gadgets containing a vulnerability. In line with the original study, we split the dataset into 80 % training and 20 % testing data, and attain a comparable accuracy.

The four systems introduced above cover the three major architecture types introduced in Section 2.2 and were published on major computer security conferences lately. It is worth mentioning that these systems provide not only a diverse view on the current use of deep learning in security but also create different challenges for the explanation methods to solve. Drebin+ and Mimicus+, for example, both make use of MLPs for detecting malware but differ greatly in the dimensionality of the input. While Mimicus+ works on a small set of engineered features, Drebin+ analyzes inputs with hundreds of thousands of dimensions that are sparsely populated over the dataset. DAMD, on the other hand, is an example of a system capable of learning from inputs of different dimensionality, ranging from 50 to more than 530,000 tokens in our experiments.

CHOICE OF EXPLANATION APPROACHES As a final step, we have to choose a set of explanation methods to evaluate throughout the chapter. Due to the diversity of the model architectures, we require the methods to be applicable to all of the systems at hand to begin with. As pointed out in Section 2.3, gradient based explanation methods, for example, are generic and applicable to all architectures relevant for our security systems. However, there exist special purpose methods like DeConvNet [318] that have been designed for CNNs specifically and are thus not suited to cover our analysis. Occlusion based approaches [e.g. 67, 318] are also difficult to apply to recurrent networks since it is unclear how tokens embedded in a vector space shall be occluded. Based on these observations we select the six explanation methods Gradients, IG, LRP, LIME, Lemna and KernelSHAP. Notice that our selection includes three white-box methods which require parameter access and three generic black-box methods based on model queries which allows us to compare the two concepts for computing explanations. Intuitively, white-box approaches should produce "better" explanations than black-box methods due to the additional information they possess about the model and we will see later that our evaluation metrics can confirm this assumption quantitatively for all security systems we evaluate.

IMPLEMENTATION DETAILS Obtaining explanations for the different network architectures and datasets is key to conduct further analyses. For our experiments we make use of the `iNNvestigate` toolbox by Alber et al. [13] that provides efficient implementations for LRP, Gradients, and IG. For the security system VulDeePecker, we use our own LRP implementation [293]. In all experiments, we set $\varepsilon = 10^{-3}$ for LRP and use $N = 64$ steps for IG. Due to the high dimensional embedding space of VulDeePecker, we increase the step count to $N = 256$ for the corresponding experiments.

We re-implement LEMNA in accordance to Guo et al. [112] in Python and use the package `cvxpy` [74] to solve the linear regression problem with Fused Lasso restriction [294]. We set the number of mixture models to $K = 3$ and the number of perturbations to $l = 500$, as suggested in the publication. The parameter S is set to 10^4 for Drebin+ and Mimicus+, as the underlying features are not sequential and to 10^{-3} for the sequences of DAMD and VulDeePecker [see 112]. Furthermore, we implement LIME with $l = 500$ perturbations and use the cosine similarity as proximity measure. For SHAP we make use of the open-source implementation by Lundberg and Lee [175] including the `KernelSHAP` solver.

3.2 Deriving Evaluation Criteria

In the following, we develop our evaluation criteria and demonstrate their utility in different examples. Before doing so, however, we address a fundamentally important question: Do the considered explanation methods provide different results at all? If the methods generated similar explanations, criteria for their comparison would be less important and any suitable method could be chosen in practice. Although Figure 1.1 indicates that explanations differ between methods, it is possible that the effect only holds for selected datapoints and vanishes when considering an entire dataset, for example.

As a first step towards comparing explanations over a dataset, we investigate the top-k features of the six explanation methods when explaining predictions of all security systems. That is, we compare the set T_i of the k features with the highest relevance from method i with the set T_j of the k features with the highest relevance from method j. In particular, we compute the *intersection size*

$$IS_k(i,j) = \frac{|T_i \cap T_j|}{k}, \tag{3.1}$$

as a measure of similarity between the two methods. The intersection size lies between zero and one, where zero indicates no overlap and one corresponds to identical top-k features.

A visualization of the intersection size averaged over the samples of three datasets is depicted in Figure 3.1. We choose $k = 10$ according to a typical use case of explainable learning: An expert investigates the top-10 features to understand a prediction. For DAMD, we use $k = 50$, to account for the long opcode sequences. We observe that the top features of the explanation methods differ considerably. For example, in the case of VulDeePecker, all methods determine different top-10 features whereas the overlap is larger for the Drebin+ dataset. For the DAMD network, the white-box and black-box explanation methods generate extremely different saliency scores. Thus, it becomes clear that explanation methods cannot be simply interchanged, and there is a need for measurable evaluation criteria.

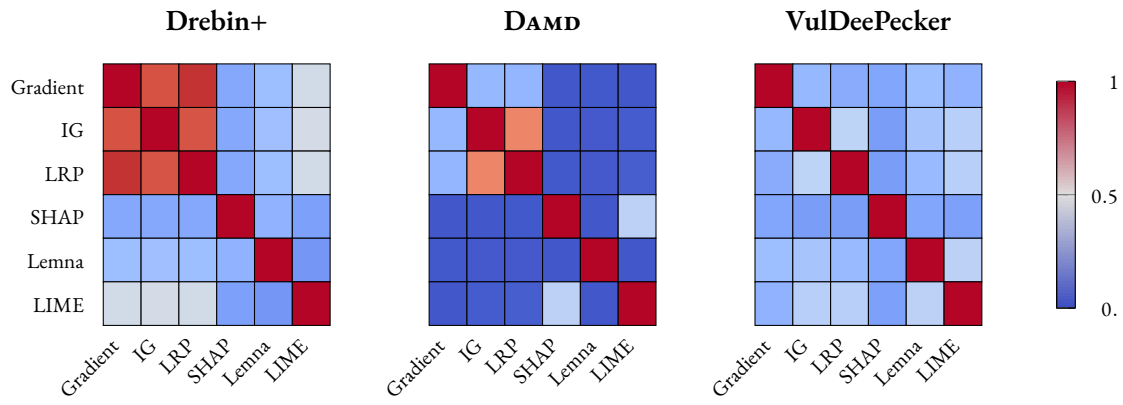


Figure 3.1: Comparison of the top-10 features for the different explanation methods using the intersection size. A white cell indicates an empty intersection set whereas a dark blue coloring indicates identical top-10 features.

To bridge the gap between deep learning in security and explanation methods developed for other domains, we divide our evaluation criteria into *general criteria* and *security criteria*. As general criteria, we consider the *descriptive accuracy* (DA) and the *descriptive sparsity* (DS) of explanations. These properties reflect how accurate and concise an explanation method captures relevant features of a prediction. Security criteria comprise the *completeness*, *stability*, *robustness*, and *efficiency* of explanation methods. These properties ensure that reliable explanations are available to a practitioner in all cases and in reasonable time, requirements that are less important in other areas of deep learning. For example, an attacker may expose pathological inputs to a security system that mislead, corrupt, or slow down the computation of explanations. In the following, we will explain each criteria in detail and discuss approaches for efficient calculation in practice.

GENERAL CRITERIA: DESCRIPTIVE ACCURACY As the first evaluation criteria, we introduce the *descriptive accuracy*. This criterion reflects the impact of features with high relevance on the prediction. As it is difficult to assess the relation between features and a prediction directly, we follow an indirect strategy and measure how removing the most relevant features changes the prediction of the neural network. The descriptive accuracy is thus related to Occlusion [69, 318] where the saliency scores act as a prior for the features to occlude. Instead of removing the feature one could also set it to a neutral baseline that depends on the classification task similar to the Integrated Gradients [266] or DeepLift [249] explanation techniques. Removing or perturbing features to measure the influence of features selected by explanation methods plays a crucial role in many evaluations and has been proposed under the name *perturbation curve* [234] or *faithfulness* [34] in other publications.

Definition 1 For a learning model f_θ, an input $\mathbf{x} \in \mathbb{R}^d$ and its explanation $\mathbf{r} \in \mathbb{R}^d$, the descriptive accuracy (DA) is calculated by removing the k most relevant features $x_1, \ldots, x_k$ according to $\mathbf{r}$ from $\mathbf{x}$, computing the new prediction using f_θ and measuring the score of the original prediction class c,

$$DA_k(\mathbf{x}, f_\theta) = f_\theta(\mathbf{x} \mid x_1 = 0, \ldots, x_k = 0)_c.$$

If we remove relevant features from a sample, the DA should decrease, as the neural network has less information for making a correct prediction. The better the explanation, the quicker the DA will drop, as the removed features capture more context of the predictions. Consequently, explanation methods with a steep decline of the descriptive accuracy provide better explanations than methods with a gradual decrease. When calculating the DA for multiple datapoints, for example each input from the training dataset, it is useful to transform the outputs of f_θ using the softmax normalization introduced in Section 2.1. This way, averaging DA scores is possible since every output value lies between zero and one with the classification result usually being close to one.

To demonstrate the utility of the descriptive accuracy, we consider a code snippet from the VulDeePecker dataset, which is shown in Figure 3.2a. The sample corresponds to a program slice and is passed to the recurrent neural network as a sequence of tokens embedded in a vector space. Figures 3.2b to 3.2d depict these tokens overlayed with the explanations of the methods LRP, IG and LIME, respectively. Recall that the VulDeePecker system truncates all code snippets to a length of 50 tokens before processing them [166], therefore some tokens from the original samples are not present in the explanations.

The code snippet shows a simple buffer overflow which originates from an incorrect calculation of the buffer size in line 7. The three explanation methods differ significantly when explaining the detection of this vulnerability. While IG highlights the `wmemset` call as important, both LRP and LIME highlight the call to `memmove` and even mark `wmemset` as speaking *against* the detection. Measuring the descriptive accuracy can help to determine which of the two explanations reflects the prediction of the system better.

General Criteria: Descriptive Sparsity Assigning high relevance to features which impact a prediction is a necessary prerequisite for good explanations. However, a human analyst can only process a limited number of these features, especially when the input is not an image and the explanation can not be processed as a saliency map visually. Thus we define the *descriptive sparsity* as a further criterion to measure the distribution of the relevance scores computed by explanation methods.

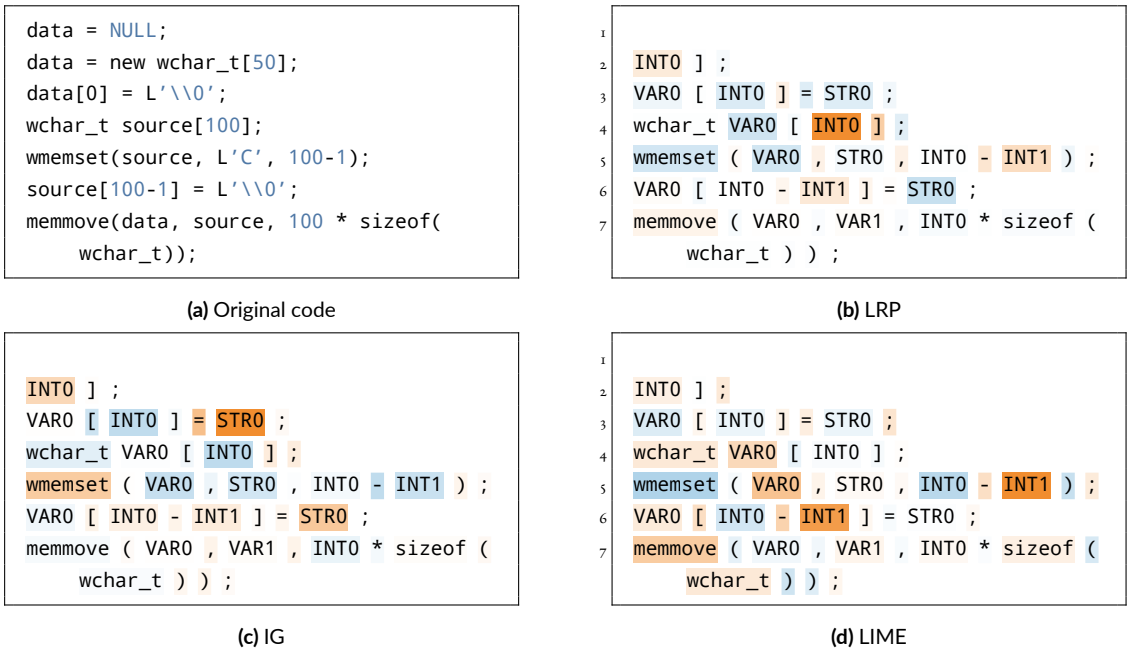


Figure 3.2: Explanations for a program slice from the VulDeePecker dataset (a) using (b) LRP, (c) IG and (d) LIME.

Definition 2 Given a learning model f_θ, an input $\mathbf{x} \in \mathbb{R}^d$ and its explanation $\mathbf{r} \in \mathbb{R}^d$, where the relevance scores have been normalized such that $-1 \leq r_i \leq 1, \forall i = 1, \ldots, d$. Denoting by h the normalized histogram over all relevance scores, the *mass around zero* (MAZ) for $t \in [0, 1]$is defined by

$$MAZ(t) = \int_{-t}^{t} h(s)\mathrm{d}s.$$

The MAZ score has to be evaluated for multiple points in $[0, 1]$ to allow for an assessment of sparsity. In our experiments, we pick 100 equidistant points $t_1, \ldots, t_{100}$ in the interval $[0, 1]$ and evaluate the MAZ score at these points. This process can be thought of as a window which starts at zero and grows uniformly into the positive and negative direction of the x axis. For each window, the fraction of relevance values that lies in the window is evaluated. Sparse explanations have a steep rise in MAZ close to zero and are flat around one, as most of the features are not marked as relevant. By contrast, dense explanations have a notable smaller slope close to zero, indicating a larger distribution of relevant features. Consequently, explanation methods with a MAZ distribution peaking at zero should be preferred over methods with less pronounced distributions. As for the descriptive accuracy, the MAZ score can be extended to reflect the sparsity of an entire dataset by computing the histogram h over all relevance scores of all points to be evaluated. The MAZ score is loosely related to the *complexity* measure proposed by Bhatt et al. [34] and to the *information gain* introduced by Bylinskii et al. [42] to measure how compact explanations are.

Table 3.2: Explanations of LRP and LEMNA for a sample of the GoldDream family from the DAMD dataset.

Id	LRP	LEMNA
0	invoke-virtual	invoke-virtual
1	move-result-object	move-result-object
2	if-eqz	if-eqz
3	const-string	const-string
4	invoke-virtual	invoke-virtual
5	move-result-object	move-result-object
6	check-cast	check-cast
7	array-length	array-length
8	new-array	new-array
9	const/4	const/4
10	array-length	array-length
11	if-ge	if-ge
12	aget-object	aget-object

As an example of a sparse and dense explanation, we consider two explanations generated for a malicious Android application of the DAMD dataset. Table 3.2 shows a snapshot of these explanations, covering opcodes of the `onReceive` method. LRP provides a crisp representation in this setting, whereas LEMNA marks the entire snapshot as irrelevant. If we normalize the relevance vectors to $[-1, 1]$ and focus on features above 0.2, LRP returns only 14 relevant features for investigation, whereas LEMNA returns 2,048 features, rendering a manual examination tedious.

In Chapter 4 we will see that the few highlighted tokens by LRP are indeed linked to the malicious behavior of the application. This example highlights the fact that the descriptive accuracy and the descriptive sparsity are not correlated and must *both* be satisfied by an effective explanation method. A method marking all features as relevant while highlighting a few ones can be accurate but is clearly not sparse. Vice versa, a method assigning high relevance to very few meaningless features is sparse but not necessarily accurate.

SECURITY CRITERIA: COMPLETENESS After introducing two generic evaluation criteria, we start focusing on aspects that are especially important for the area of computer security. In a security system, an explanation method must be capable of creating proper results in all possible situations. If some inputs, such as pathological data or corner cases, cannot be processed by an explanation method, an adversary may trick the method into producing *degenerated* results of no use for the practitioner. Consequently, we propose *completeness* as the first security-specific criterion.

Definition 3 An explanation method is complete, if it can generate non-degenerated explanations for all possible input vectors of the learning model f_θ. An explanation is called degenerated if it was generated from an ill-posed optimization problem resulting in saliency values like zero, `NaN` or a constant value for all features.

As an example of this problem, Table 3.3 shows explanations generated by the methods Gradients and SHAP for a benign Android application of the Drebin+ dataset. The Gradients explanation highlights the `touchscreen` feature in combination with the `launcher` category and the `internet` permission as an explanation for the benign classification. SHAP, however, creates an explanation of zeros which provides no insights. The reason for this degenerated explanation is rooted in the random perturbations used by SHAP. By flipping the value of features, these perturbations aim at changing the class label of the input. However, as there exist far more benign features than malicious ones in the case of Drebin+, setting malicious features to zero for a perturbation usually leads to a benign classification but setting benign features to zero, often does not impact the classification result. As a consequence, the linear regression problem in Equation (2.13) becomes ill-posed and results in a degenerated explanation of no meaningfulness.

Most white-box methods are complete by definition, as they calculate relevance vectors directly from the weights of the neural network. For black-box methods, however, the situation is different. Since the majority of these approaches is based on surrogate models, approximating the prediction function f_θ by random perturbations is essential. However, depending on the feature distribution and dimensionality of the input space, these approaches can return degenerated explanations. We investigate this phenomenon in greater detail when evaluating the stability criterion later.

Table 3.3: Explanations for a benign Android application generated for Drebin+ using Gradients and SHAP.

Id	Gradients	SHAP
0	`feature::android.hardware.touchscreen`	`feature::android.hardware.touchscreen`
1	`intent::android.intent.category.LAUNCHER`	`intent::android.intent.category.LAUNCHER`
2	`real_permission::android.INTERNET`	`real_permission::android.INTERNET`
3	`api_call::android/webkit/WebView`	`api_call::android/webkit/WebView`
4	`intent::android.intent.action.MAIN`	`intent::android.intent.action.MAIN`
5	`url::translator.worldclockr.com`	`url::translator.worldclockr.com`
6	`permission::android.permission.INTERNET`	`permission::android.permission.INTERNET`
7	`activity::.Main`	`activity::.Main`

Security Criteria: Stability In addition to complete results, the explanations generated in a security system need to be reliable. That is, relevant features must not be affected by fluctuations and need to remain stable over time in order to be useful for an expert. As a consequence, we define *stability* as another security-specific evaluation criterion.

Definition 4 An explanation methods is stable, if the generated explanations do not vary between multiple runs. That is, for any run i and j of the method, the intersection size of the top k features T_i and T_j should be close to 1, that is, $IS_k(i,j) > 1-\varepsilon$ for some small threshold ε and for all $k \leq d$.

The stability of an explanation method can be empirically determined by running the methods multiple times and computing the average intersection size, as explained in the beginning of this chapter. White-box methods are deterministic by construction since they perform a fixed sequence of computations for generating an explanation. Most black-box methods, however, require random perturbations to compute their output which can lead to different results for the same input. Table 3.4, for instance, shows the output of LEMNA for a PDF document from the Mimicus+ dataset over two runs. Some of the most relevant features from the first run receive very little relevance in the second run and vice versa, rendering the explanations unstable.

Table 3.4: Two explanations from LEMNA for the same example from the Mimicus+ dataset computed in different runs.

Id	LEMNA (Run 1)	LEMNA (Run 2)
0	pos_page_min	pos_page_min
1	count_js	count_js
2	count_javascript	count_javascript
3	pos_acroform_min	pos_acroform_min
4	ratio_size_page	ratio_size_page
5	pos_image_min	pos_image_min
6	count_obj	count_obj
	...	...
27	pos_image_max	pos_image_max
28	count_page	count_page
29	len_stream_avg	len_stream_avg
30	pos_page_avg	pos_page_avg
31	count_stream	count_stream
32	moddate_tz	moddate_tz
33	len_stream_max	len_stream_max
34	count_endstream	count_endstream

Security Criteria: Efficiency When operating a security system in practice, explanations need to be available in reasonable time. While low run-time is not a strict requirement in general, time differences between minutes and milliseconds are still significant. For example, when dealing with large amounts of data, it might be desirable for the analyst to create explanations for every sample of an entire class. We thus define *efficiency* as a further criterion for explanation methods in security applications.

Definition 5 We consider a method efficient if it enables providing explanations without delaying the typical workflow of the practitioner.

As the workflow depends on the particular security task, we do not define concrete run-time numbers, yet we provide a negative example to illustrate consequences for the practitioner when using slow explanation methods. The run-time of the method LEMNA depends on the size of the inputs. For the largest sample of the Damd dataset with 530,000 features, it requires about one hour for computing an explanation, which obstructs the workflow of inspecting Android malware severely.

Security Criteria: Robustness As the last criterion, we consider the *robustness* of explanation methods to attacks. As described in Section 2.4, there exist multiple attack approaches [e.g. 77, 255, 321] that demonstrate the vulnerability of explanation methods towards adversarial perturbations tricking them into returning incorrect relevance vectors, similar to the phenomenon of adversarial examples [45, 269]. The objective of these attacks is to disconnect the explanation from the underlying prediction, such that arbitrary relevance values can be generated that do not explain the behavior of the model.

Definition 6 An explanation method is robust if the computed relevance vector can not be decoupled from the prediction by an adversarial perturbation and if the prediction cannot be changed without changing the explanation severely at the same time.

Clearly, it is difficult to measure robustness towards existing attacks quantitatively and the robustness of explanation methods is an ongoing research field. At the time of this writing, only few defenses have been proposed in the literature. Rieger and Hansen [224] propose the usage of ensembles of multiple explanations to impede simple perturbation attacks. Lu et al. [174] and Dombrowski et al. [77] show that the high dimensional and non-smooth decision boundary of modern neural networks is the root cause rendering attacks possible. For evasion they propose smoothing the decision function via regularization or special non-linearities. To this end, we assess the robustness of the explanation methods based on the existing literature.

We start our evaluation by measuring the descriptive accuracy (DA) of the explanation methods as defined in Section 3.2. In particular, we successively remove the most relevant features from the samples of the datasets and measure the decrease in the classification score. For Drebin+ and Mimicus+, we remove features by setting the corresponding dimensions to zero. For Damd, we replace the most relevant instructions with the no-op opcode, and for VulDeePecker we substitute the tokens to be removed with an embedding-vector of zeros.

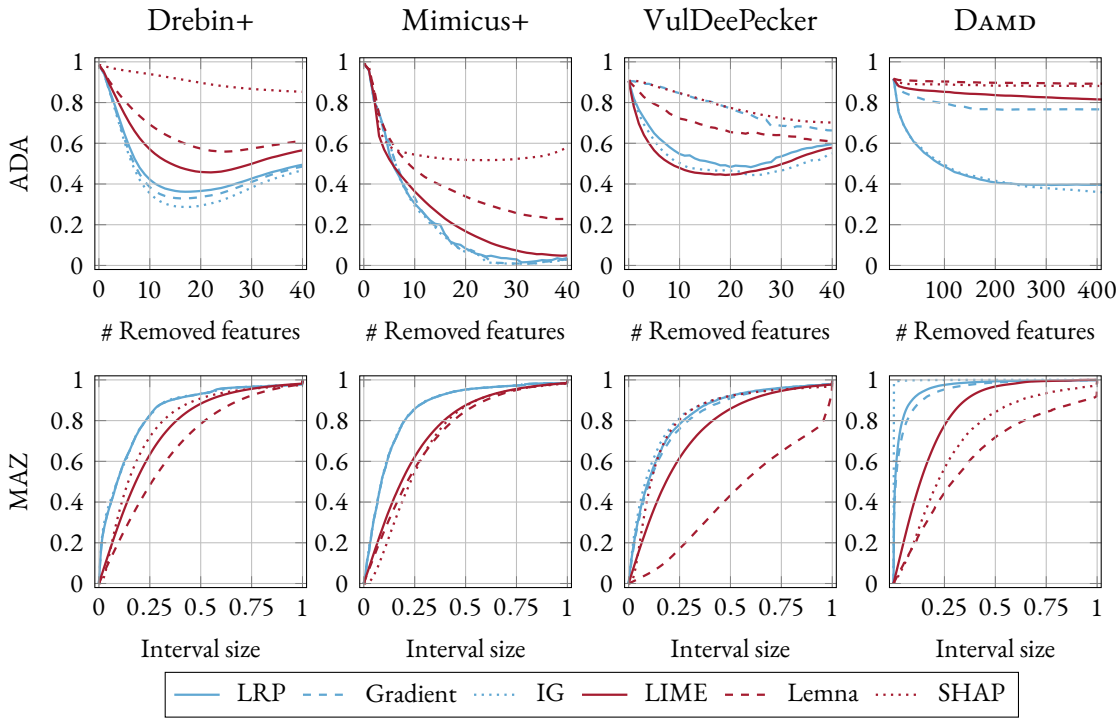


Figure 3.3: Descriptive accuracy and sparsity for the considered explanation methods. Top row: Average descriptive accuracy (ADA); bottom row: Sparsity measured as mass around zero (MAZ).

The top row in Figure 3.3 shows the results of this experiment. As the first observation, we find that the DA curves vary significantly between the explanation methods and security systems. However, the methods IG and LRP consistently obtain strong results in all settings and show steep declines of the descriptive accuracy already for few removed features. Only on the VulDeePecker dataset, the black-box method LIME can provide explanations with comparable accuracy. Notably, for the Damd dataset, IG and LRP are the only methods to generate real impact on the outcome of the classifier. For Mimicus+, IG, LRP and Gradients achieve a perfect accuracy decline after only 25 features and thus the white-box explanation methods outperform the black-box methods in this experiment.

Table 3.5(a) shows the *area under curve* (AUC) for the descriptive accuracy curves from Figure 3.3 with the best method for each dataset highlighted in red. We observe that IG is the best method over all datasets—lower values indicate better explanations—followed by LRP. In comparison to other methods it is up to 48 % better on average. Intuitively, this considerable difference between the white-box and black-box methods makes sense, as white-box approaches can utilize internal information of the neural networks that are not available to black-box methods.

Table 3.5: Descriptive accuracy (DA) and sparsity (MAZ) for the different explanation methods.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	0.580	0.257	0.919	0.571
LEMNA	0.656	0.405	0.983	0.764
SHAP	0.891	0.565	0.966	0.869
Gradients	0.472	0.213	0.858	0.856
IG	0.446	0.206	0.499	0.574
LRP	0.474	0.213	0.504	0.625

(a) Area under the DA curves from Figure 3.3.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	0.757	0.752	0.833	0.745
LEMNA	0.681	0.727	0.625	0.416
SHAP	0.783	0.716	0.713	0.813
Gradients	0.846	0.856	0.949	0.816
IG	0.847	0.858	0.999	0.839
LRP	0.846	0.856	0.964	0.827

(b) Area under MAZ curves from Figure 3.3.

DESCRIPTIVE SPARSITY We proceed by investigating the sparsity of the generated explanations with the MAZ score defined in Section 3.2. The second row in Figure 3.3 shows the result of this experiment for all datasets and methods. We observe that the methods IG, LRP, and Gradients show the steepest slopes and assign the majority of features little relevance, which indicates a sparse distribution. By contrast, the other explanation methods provide flat slopes of the MAZ close to zero, as they generate relevance values with a broader range and thus are less sparse.

For Drebin+ and Mimicus+, we observe an almost identical level of sparsity for LRP, IG and Gradients supporting the findings from Figure 3.1. Interestingly, for VulDeePecker, the MAZ curve of LEMNA shows a strong increase close to one, indicating that it assigns high relevance to a lot of tokens. While this generally is undesirable, in case of LEMNA, this is

founded in the basic design and the use of the Fused Lasso constraint. In case of DAMD, we see a massive peak at zero for IG, showing that it marks almost all features as irrelevant. According to the previous experiment, however, it simultaneously provides a very good accuracy on this data. The resulting sparse and accurate explanations are particularly advantageous for a human analyst since the DAMD dataset contains samples with up to 530,000 features. The explanations from IG therefore provide a compressed yet accurate representation of the sequences which can be inspected easily.

We summarize the performance on the MAZ metric by calculating the *area under curve* and report it in Table 3.5(b). A high AUC indicates that more features have been assigned a relevance close to zero, that is, the explanation is more sparse. We find that the best methods again are white-box approaches, providing explanations that are up to 50 % sparser compared to the other methods in this experiment.

COMPLETENESS OF EXPLANATIONS We further examine the completeness of the explanations. As we have seen in Table 3.3, some explanation methods can not calculate meaningful relevance values for all inputs. In particular, perturbation-based methods suffer from this problem, since they determine a regression with labels derived from random perturbations. To investigate this problem, we monitor the creation of perturbations and their labels for the different datasets.

When creating perturbations for some sample $\mathbf{x}$ it is essential for black-box methods that a fraction $0 \leq p \leq 1$ of them is classified as belonging to the opposite class of $\mathbf{x}$. In an optimal case one can achieve $p \approx 0.5$, however during our experiments we find that $p = 0.05$ can

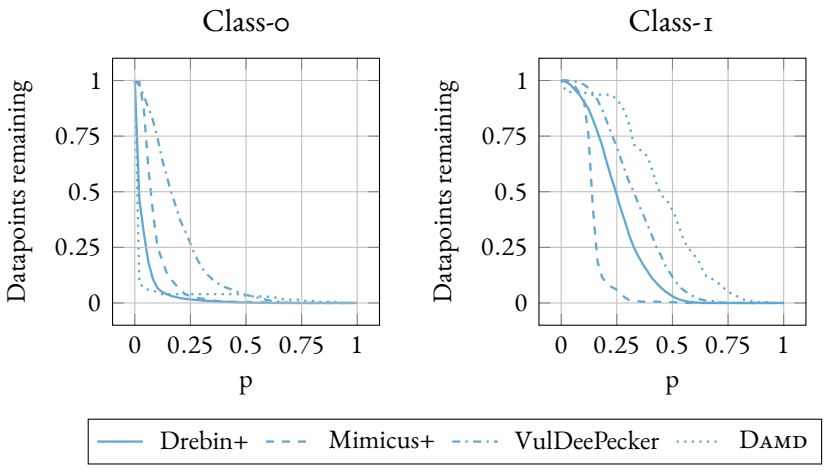


Figure 3.4: Perturbation label statistics of the datasets. For each percentage of perturbations from the other class the percentage of samples achieving this number is shown.

sometimes be sufficient to calculate a non-degenerated explanation. Figure 3.4 shows for each value of p and all datasets the fraction of samples remaining when enforcing a percentage p of perturbations from the opposite class.

In general, we observe that creating malicious perturbations from benign samples is a hard problem, especially for Drebin+ and DAMD. For example, in the Drebin+ dataset only 31 % of the benign samples can obtain a p value of 5 % which means that more than 65 % of the whole dataset suffer from degenerated explanations. Table 3.6 shows a concrete example of the dataset statistics when enforcing 5 % of the labels generated by the perturbations to be from the opposite class. On average, 33 % of the samples cannot be explained well, as the computed perturbations contain too few instances from the opposite class. In particular, we observe that creating malicious perturbations from benign samples is a hard problem in the case of Drebin+ and DAMD, where only 24.2 % and 5.9 % of the benign samples achieve sufficient perturbations from the opposite class respectively.

Table 3.6: Incomplete explanations of black-box methods. First two columns: Samples remaining when enforcing at least 5 % perturbations of opposite class. Final column: Total percentage of incomplete explanations in the dataset.

System	Class-0	Class-1	Incomplete
Drebin+	24.2%	97.1 %	66.3%
Mimicus+	73.5 %	98.9%	14.2%
VulDeePecker	90.5 %	99.8 %	7.1 %
DAMD	5.9%	94.8 %	44.9%
Average	48.3 %	97.7 %	33.2%

The problem of incomplete explanations is rooted in the imbalance of features characterizing malicious and benign data in the datasets. In the case of Drebin+, only few features make a sample malicious but there exists a large variety of features turning a sample benign. As a consequence, randomly setting malicious features to zero leads to a benign classification, while setting benign features to zero usually does not impact the prediction. For DAMD, in a similar manner, we have very long sequences of opcodes where malicious behavior is defined by a specific sequence of tokens. Generating these specific instructions randomly is extremely unlikely. Therefore, it is often not possible to explain predictions for benign applications limiting the applicability of black-box explanation methods for the practitioner.

In summary, we argue that perturbation-based explanation methods should only be used in security settings where incomplete explanations can be compensated by other means. In all other cases, one should prefer the usage of white-box methods in the context of security.

Stability of Explanations We proceed to evaluate the stability of the explanation methods when processing inputs from the four security systems. To this end, we apply the explanations to the same samples over multiple runs and measure the average intersection size between the runs.

Table 3.7 shows the average intersection size between the top k features for three runs of the methods as defined in Equation (3.1). We use $k = 10$ for all datasets except for DAMD where we use $k = 50$ due to the larger input space. Since the outputs of Gradients, IG, and LRP are deterministic, they reach the perfect score of 1.0 in all settings and thus do not suffer from limitations concerning stability. For the perturbation-based methods, however, stability poses a severe problem since none of those methods obtains a intersection size of more than 0.5. This indicates that on average half of the top features do not overlap when computing explanations on the same input. Furthermore, we see that the assumption of *locality* of the perturbation-based methods does not apply for all models under test, since the output is highly dependent on the perturbations used to approximate the decision boundary. Therefore, the best methods for the stability criterion beat the perturbation-based methods by a factor of at least 2.5 on all datasets.

Table 3.7: Average intersection size between top features for multiple runs. Values close to one indicate greater stability.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	0.480	0.446	0.040	0.446
LEMNA	0.4205	0.304	0.016	0.416
SHAP	0.257	0.411	0.007	0.440
Gradients	1.000	1.000	1.000	1.000
IG	1.000	1.000	1.000	1.000
LRP	1.000	1.000	1.000	1.000

Efficiency of Explanations We finally examine the efficiency of the different explanation methods for the security systems. Our experiments are performed on a regular server system with an `Intel Xeon E5 v3` CPU at 2.6 GHz. It is noteworthy that the methods Gradients, IG and LRP can benefit from computations on a graphical processing unit (GPU), therefore we report both results but use only the CPU results to achieve a fair comparison with the black-box methods.

Table 3.8 shows the average run-time per input for all explanations methods and we observe that Gradients and LRP achieve the highest throughput in general beating the other methods by orders of magnitude. This advantage arises from the fact that data can be processed *batch-wise* for methods like Gradients, IG, and LRP, that is, explanations can be cal-

Table 3.8: Run-time per sample in seconds. Note the range of the different times from microseconds to minutes.

Method	Drebin+	Mimicus+	DAMD	VulDeePecker
LIME	3.1×10^{-2}	2.8×10^{-2}	7.4×10^{-1}	3.0×10^{-2}
LEMNA	4.6	2.6	6.9×10^{2}	6.1
SHAP	9.1	4.3×10^{-1}	4.5	5.0
Gradients	8.1×10^{-3}	7.8×10^{-6}	1.1×10^{-2}	7.6×10^{-4}
IG	1.1×10^{-1}	5.4×10^{-5}	6.9×10^{-1}	4.0×10^{-1}
LRP	8.4×10^{-3}	1.7×10^{-6}	1.3×10^{-2}	2.9×10^{-2}
GPU	**Drebin+**	**Mimicus+**	**DAMD**	**VulDeePecker**
Gradients	7.4×10^{-3}	3.9×10^{-6}	3.5×10^{-3}	3.0×10^{-4}
IG	1.5×10^{-2}	3.9×10^{-5}	3.0×10^{-1}	1.3×10^{-1}
LRP	7.3×10^{-3}	1.6×10^{-6}	7.8×10^{-3}	1.1×10^{-2}

culated for a set of samples at the same time. The Mimicus+ dataset, for example, can be processed in one batch resulting in a speed-up factor of more than 16,000× over the fastest black-box method. In general we note that the white-box methods Gradients and LRP achieve the fastest run-time since they require a single backwards-pass through the network. IG, in contrast, requires multiple gradient evaluations and thus multiple backwards-passes increasing the runtime significantly. Moreover, computing these methods on a GPU results in additional speedups of a factor up to three.

The run-time of the black-box methods increases for high dimensional datasets, especially DAMD, since the regression problems need to be solved in higher dimensions and requires far more perturbations. While the speed-up factors are already enormous, we have not even included the creation of perturbations and their classification, which consume additional run-time as well.

ROBUSTNESS OF EXPLANATIONS In Section 2.4 we introduced multiple approaches to attack the outcome of explanation methods including adversarial perturbations [77, 321] or minimal changes to the model that can decouple the explanation from the true behavior [17, 255]. These attacks are possible due to the high dimensional and irregular decision boundary of neural networks [77, 174] and thus cannot always be prevented. Despite the presence of defense approaches [224], we conclude that both white-box and black-box explanation methods are not robust and vulnerable. Still, these powerful attacks require access to specific parts of the victim system as well as further extensions to work in discrete domains. These attack vectors should be considered and prevented whenever using explanation methods in security applications.

Summary A strong explanation method is expected to achieve good results for each criterion and on each dataset. For example, we have seen that the Gradients method computes sparse results in a decent amount of time. The features, however, are not accurate on the Damd and VulDeePecker dataset. Equally, the relevance values of SHAP for the Drebin+ dataset are sparser than those from LEMNA but suffer from instability. To provide an overview, we average the performance of all methods over the four datasets and summarize the results in Table 3.9.

Table 3.9: Results of the evaluated explanation methods. The last column summarizes these metrics in a rating comprising three levels: strong(●), medium (◒), and weak (○).

Method	Accuracy	Sparsity	Completeness	Stability	Efficiency	Robustness	Overall Rating
LIME	0.582	0.772	–	0.353	2.1×10^{-1} s	○	◒◒○○◒○
LEMNA	0.702	0.612	–	0.289	1.8×10^{2} s	○	○○○○○○
SHAP	0.823	0.757	–	0.279	4.8 s	○	○◒○○○○
Gradients	0.600	0.867	✓	1.000	5.0×10^{-3} s	○	◒●●●●○
IG	0.431	0.886	✓	1.000	3.0×10^{-1} s	○	●●●●◒○
LRP	0.454	0.873	✓	1.000	5.0×10^{-2} s	○	●●●●●○

For each of the six evaluation criteria, we assign each method one of the following three categories: ●, ◒, and ○. The ● category is given to the best explanation method and other methods with a similar performance. The ○ category is assigned to the worst method and methods performing equally bad. Finally, the ◒ category is given to methods that lie between the best and worst methods.

Based on Table 3.9, we can see that white-box explanation methods achieve a better ranking than black-box methods in all evaluation criteria. Due to the direct access to the parameters of the neural network, these methods can better analyze the prediction function and are able to identify relevant features. In particular, IG and LRP are the best methods overall regarding our evaluation criteria. They compute results in less than 50 ms in our benchmark, mark only few features as relevant, and the selected features have great impact on the decision of the classifier. These methods also provide deterministic results and do not suffer from incompleteness. As a result, we recommend to use these methods for explaining deep learning in security.

In general, whether white-box or black-box methods are applicable also depends on *who* is generating the explanations: If the developer of a security system wants to investigate its prediction, direct access to all model parameters is typically available and white-box methods can be applied easily. Similarly, if the learning models are shared between practitioners, white-box approaches are also the method of choice. If the learning model, however, is trained by a remote party, such as a machine-learning-as-a-service providers, only black-box methods are

applicable. Likewise, if an auditor or security tester inspects a proprietary system, black-box methods also become handy, as they do not require reverse-engineering and extracting model parameters. For these cases, we recommend the black-box method LIME as it shows the best performance in our experiments overall. As another remedy, it is also possible to apply a *model stealing* attack [e.g. 280] on the service that aims to create a copy of the model that closely resembles its performance and properties. In the easiest case, a suitable training set is available and the labels can be obtained by querying the black-box model allowing to start a regular training process. Once such a shadow model is obtained, white-box methods can be used to generate explanations and we will evaluate whether the performance is compatible in the following paragraph.

Model Stealing for White-Box Explanations To evaluate the differences between the explanations of surrogate models to the original ones we conduct an experiment on the Drebin+ and Mimicus+ datasets as follows: We use the predictions of the original model from Grosse et al. [106] which has two dense layers with 200 units each and use these predictions to train three surrogate models. The number of layers is varied to be $[1, 2, 3]$ and the number of units in each layer is always 200 resulting in models with higher, lower and the original complexity. For each model we calculate explanations via LRP and compute the intersection size given by Equation (3.1) for $k = 10$.

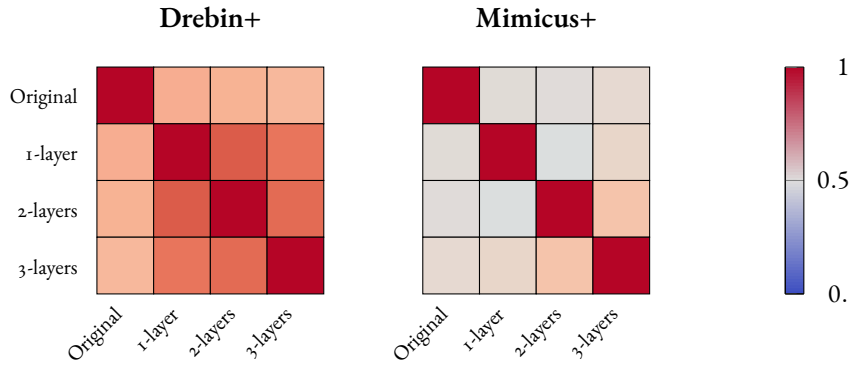


Figure 3.5: Intersection size of the Top-10 features of explanations obtained from models that were stolen from the original model of the Drebin+ and Mimicus+ dataset.

The results in Figure 3.5 show that the models deliver similar explanations to the original model ($IS_{10} \approx 0.7$) although having different architectures for the Drebin+ dataset. However, the similarity between the stolen models is clearly higher ($IS_{10} \approx 0.85$). For the Mimicus+ dataset, we observe a general stability of the learned features at a lower level ($IS_{10} \approx 0.55$). These results indicate that the explanations of the stolen models are better

than those obtained from black-box methods (see Figure 3.1) but still deviate from the original model, i.e., there is no transferability between the explanations. At all, model stealing can be considered a good alternative to the usage of black-box explanation methods.

3.4 Related Work

The appearance of explanation methods quickly raised the questions of how they can be compared and what the differences between them constitute of [98, 323]. Our measures intersection size, sparsity or stability have been proposed under different names to evaluate explanation methods [34, 68, 155] and implemented in software solutions [10]. Other measures include the crafting of adversarial examples based on relevant features from explanation algorithms [124] or comparing saliency scores of a network trained with foreground labels with one trained for the background objects [313]. Building on a debugging approach with explanations, Idahl et al. [132] propose to insert artificial decision rules, so called *decoys*, into a dataset and compare which methods mark them as important.

In terms of generalization Yeh et al. [315] introduce the *infidelity* measure and show that approaches like IG or LRP are optimal in this light. Similarly, Wang et al. [292] summarize existing evaluation metrics and categorize them into the logical concepts necessity and sufficiency. They seek a minimal set of pixels that is sufficient for a classification and a smallest set of pixels required to change a classification and compare different approaches towards these metrics. It turns out that the majority of methods only optimizes one of the concepts and thus endorsing a single method for interpretation is difficult. Han et al. [116] group explanation methods using the concept of function approximations from calculus and show that different explanation methods optimize different distance measures in this viewpoint. Ancona et al. [16] discuss similarities between gradient based explanation methods and find that LRP and Gradients are identical, for networks with linear layers and ReLu non-linearities, for example. The *n-sensitivity* metric is introduced to evaluate whether the removal of features has linear impact on the classification score and is thus related to descriptive accuracy. Removing or perturbing important features plays a key role in many evaluation approaches [65] under names like *faithfulness* [34] or *perturbation curve* [234]. Such approaches have also been extended to removing entire regions of an image using image segmentation algorithms, for example [25, 225]. Hooker et al. [122] point to a potential limitation of removal strategies by showing that after removing a large portion of features from a sample, the classification will change since the learning model has not seen inputs of this distribution in the training process. Consequently, the explanations should be ranked by the loss in classification performance after *retraining* the model without the relevant features instead.

4

From Explanations to Security Insights

The evaluation performed in Chapter 3 allows the developer to select a suitable explanation method for generating saliency scores of the machine learning model at hand. Due to access to the model and training data, he will likely use a white-box approach that enhances the training dataset D by a set of *relevance vectors* containing an explanation for each datapoint in D under the model f_θ. If the training dataset is large, however, he can not investigate every explanation manually and requires a curator that presents interesting explanations to him in an intelligent fashion. In Section 4.1 we present the PROF selection scheme that allows the practitioner to query prototypical and unique explanations from the corpus of relevance vectors. Based on this scheme we collect insights into the neural networks from the previous chapter in Section 4.2. We will see that the explanations reveal various shortcomings in the learning models and datasets indicating that they are not ready for usage in practical applications yet. Next, we present a novel approach to vet malware tags in the context of dynamical malware analysis in Section 4.3. Finally, we review related work on the usage of explanation methods for analyzing learning models in security in Section 4.4.

4.1 PROF: A FRAMEWORK FOR SELECTING EXPLANATIONS

Leveraging explanation methods for learning models in security applications enhances the training dataset D by a set of *relevance vectors* $\mathcal{R} = \mathbf{r}_1, \ldots, \mathbf{r}_n$ where $\mathbf{r}_i \in \mathbb{R}^d$ represents the explanation for the training datapoint $\mathbf{x}_i$ under the model f_θ. The practitioner is now confronted with the analysis of these explanations in order to obtain insights to the model and dataset at hand. A straight forward solution would be the usage of clustering algorithms, like k-means-clustering [171, 177], that partition $\mathcal{R}$ into groups where the intra-group similarity is high but the inter-group similarity is low [18, 158]. However, these approaches require the practitioner to select the number of groups k, which is likely unknown in advance, and deliver only prototypical explanations corresponding to the centroids of the clusters. Explanations from sparsely populated areas, however, can be of similar value since they may indicate spurious correlations, wrongly labeled datapoints or invalid features. Therefore, we propose a selection scheme that picks prototypical and unique explanations from $\mathcal{R}$ and allows the practitioner to dynamically select as many explanations as he desires from both groups. We build the scheme based on three desiderata we believe are advantageous when investigating a large corpus of explanations for a machine learning based security system:

1. **Prototype and outlier identification** There are two groups of explanations that are of interest for the practitioner. The first group comprises prototypical explanations that serve as representatives for a broader set of similar examples in $\mathcal{R}$. These explanations enable users to examine a larger collection of similar instances in a single review. Conversely, unique or anomalous outlier explanations also give valuable insights by potentially highlighting datapoints with defects, like incorrect labels or invalid features. Therefore, our framework should be capable of categorizing explanations into either of these classes and enable the practitioner to request datapoints from both categories.

2. **Variable selection size** Many clustering algorithms require the desired number of groups k as an input parameter. However, the practitioner usually does not know this number in advance and is thus required to perform an extensive parameter search with multiple runs of the clustering algorithm and repetitive analysis steps afterwards. We therefore opt for a selection algorithm that allows the user to query as many examples as he desires in the inspection process without knowing this number before. This property ensures that the practitioner can stop the selection process when he has a good feeling about the explanations in the dataset or encounters examples similar to previously seen ones.

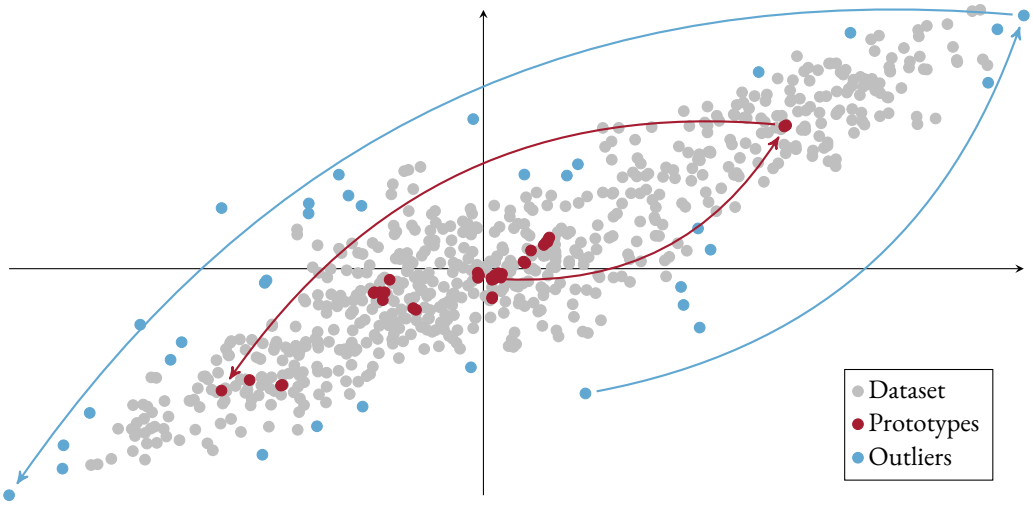


Figure 4.1: Overview of the PROF scheme: The prototypes (red) and outliers (blue) are filtered based on a similarity score and visited by the concept of a farthest-first-traversal.

3. **Maximal selection difference** Analyzing explanations is tedious work, therefore the practitioner would like to inspect as few explanations as possible while still getting an overview over the explanation corpus $\mathcal{R}$. To this end, it is important that the selected explanations are different to each other such that no redundant work is performed in the inspection process.

To fulfil the criteria mentioned above, we propose the PROF scheme based on **pr**ototypes, **o**utliers and **f**arthest-first traversal of explanations. The scheme consists of three steps, namely *scoring*, *explanation filtering* and *traversal* which we will describe in detail in the following. Figure 4.1 visualizes each step of the PROF scheme for 200 datapoints with $k = 5$ and can ease the understanding of the underlying concept.

SCORING In the first step of the selection scheme, we require a measure indicating whether an explanation is rather prototypical or unique. To this end, we rely on the concepts proposed by Harmeling et al. [117] that assign such score to a datapoint based on the distance to its nearest neighbors in the underlying dataset. Concretely, for a given relevance vector $\mathbf{r} \in \mathcal{R}$ we denote by $\eta_1(\mathbf{r}), \ldots, \eta_k(\mathbf{r})$ the k nearest neighbors of $\mathbf{r}$, i.e. the points in $\mathcal{R}$ with the smallest Euclidean distance to $\mathbf{r}$. Based on these neighbors, Harmeling et al. [117] define the three measures κ, γ and δ given by

$$\kappa(\mathbf{r}) = \|\mathbf{r} - \eta_k\|_2, \qquad \gamma(\mathbf{r}) = \frac{1}{k}\sum_{l=1}^{k}\|\mathbf{r} - \eta_l\|_2, \qquad \delta(\mathbf{r}) = \left\|\frac{1}{k}\sum_{l=1}^{k}\mathbf{r} - \eta_l\right\|_2. \tag{4.1}$$

The metric κ simply measures the distance to the k-th nearest neighbor whereas γ represents the average distance to the nearest neighbors. δ also takes the direction of the differences into account and computes the length of the mean of the vectors pointing from $\mathbf{r}$ to its k nearest neighbors. All of the metrics assign prototypical points, i.e. points in regions with high population of other datapoints, small scores whereas outliers residing far away from other datapoints receive a large score. Computing a score from above for each point in $\mathcal{R}$ allows to re-order the explanations in ascending order and thereby ranking them from outliers to prototypes, fulfilling the first requirement from above. While each measure has its advantages and disadvantages [see 117, for a discussion] we will use γ in the following since it is the most intuitive one for a human analyst.

Data filtering Assigning each explanation a score from above creates a score distribution over the set $\mathcal{R}$ and we are interested in the tails of the distribution where prototypes and outliers reside. If μ is the mean score over $\mathcal{R}$, we define the tails by a parameter $p \in [0, 1]$ by calculating the confidence interval $I = [\mu - a, \mu + a]$ around μ such that a fraction p of all scores falls into I. Then, a prototype is an explanation with a score smaller than $\mu - a$ and an outlier is an explanation with a score larger than $\mu + a$. In Figure 4.1 we use $p = 0.9$ and highlight prototypes (red) and outliers (blue) resulting from this choice. It becomes apparent that the prototypical points come from regions with high density whereas the outliers are points close to the convex hull of the datapoints.

Traversal Equipped with a set of prototypes and outliers we have to pick a traversal scheme to present the explanations to the practitioner. To achieve maximal selection difference as defined above, we leverage the *farthest-first-traversal* [102, 230] that defines a traversal over the prototypes and outliers in the following way: We pick the most prototypical explanation (prototype traversal) or the most unique explanation (outlier traversal) and as a starting point $\mathbf{r}^{(0)}$ and add $\mathbf{r}^{(0)}$ to the set of visited points V. If another request from the practitioner arrives, we return the explanation with the largest distance to all points in V, i.e.

$$\mathbf{r}^{(t+1)} = \mathbf{r}_i, \qquad \text{where} \qquad i = \max_{\substack{j=1,\dots,n \\ \mathbf{r}_j \notin V}} \sum_{l=0}^{t} \|\mathbf{r}_j - \mathbf{r}^{(l)}\|_2. \tag{4.2}$$

The first three steps of the prototype- and outlier traversal are depicted by arrows in red and blue respectively in Figure 4.1. By construction, the Prof scheme selects new explanations that are maximally different to the explanations obtained so far while the data filtering ensures that only interesting points at the tails of the score distribution are visited at all.

In terms of complexity, the PROF scheme has to determine the k nearest neighbors of every datapoint to compute the scores in Equation (4.1) which requires calculating the distance matrix of $\mathcal{R}$ and thus $\mathcal{O}(n^2 d)$ operations. For γ we can select the k nearest neighbors in $\mathcal{O}(nk)$ for a single datapoint and thus end up with a total complexity of $\mathcal{O}(n^2 d + n^2 k)$ for the score calculation. Sorting a row in the distance matrix in $\mathcal{O}(n \log(n))$ accelerates the selection process and results in a complexity of $\mathcal{O}(n^2 d + n^2 \log(n))$. Another advantage of this setup is that testing different values for k, which is the most important hyper-parameter in the algorithm, does not cause further costs. Having a distance matrix available, the farthest first traversal simply requires to sum the rows corresponding to the datapoints in V and finding the maximum with a linear lookup, i.e. the complexity for t traversal steps is given by $\mathcal{O}(tn)$.

4.2 Security Model Analysis

Equipped with the PROF scheme to select interesting explanations we can continue the analysis of the learning models from Chapter 3. To this end, we use the explanations generated by the LRP approach due to its excellent performance and low run-time and apply the PROF scheme to the explanations of the training data of all datasets with parameters $k = 10$ and $p = 0.9$. As a first step, we qualitatively assess the explanations and then perform further experiments to understand the learning model and the features used for classification better.

4.2.1 Malware Detection

The majority of networks used for our experiments in Chapter 3 were trained for the detection of malicious PDF documents or Android applications. We will start our analysis with these models in the following subsection.

PDF Malware classification Figure 4.2 (left two columns) shows the first two examples selected by the PROF scheme when applied to explanations of the malicious documents of the Mimicus+ dataset. When inspecting the explanations we firstly observe that our selection scheme fulfils the requirements defined in Section 4.1 very well since the set of important features is disjoint between the different examples. Inspecting the explanations, we observe that the `count_javascript` and `count_js`, which both stand for the number of JavaScript elements in the document, are important. The strong impact of these elements is meaningful, as JavaScript is frequently used in malicious PDF documents [142] and can also be embedded in acroform markers indicated by the `count_acroform` feature. However, we also identify features in the explanations that are non-intuitive. For example, features like `count_trailer`

Id	Prototype	Prototype	Outlier	Outlier
0	count_javascript	count_acroform	delta_tz	moddate_mismatch
1	count_js	subject_lc	pdfid1_num	count_objstm
2	count_trailer	author_lc	moddate_tz	createdate_mismatch
3	count_endobj	title_lc	pdfid0_uc	pos_page_max
4	...	...	...	...
5	count_eof	count_eof	count_page	creator_mismatch
6	count_box_other	pos_eof_min	count_box_other	pos_image_min
7	pos_eof_min	len_stream_max	createdate_tz	pdfid0_num
8	len_stream_max	count_font	pos_page_max	moddate_tz

Figure 4.2: Prototypes (left two columns) and outliers (right two columns) of malicious PDF documents from the Mimicus+ dataset as presented by the first two steps of the Prof scheme.

that measures the number of trailer sections in the document or `subject_lc` that counts the number of lowercase letters in the subject of the document can hardly be related to security and rather constitute artifacts in the dataset captured by the learning process. Such features with no malware context can also be found in the outlier explanations (right two columns) and are like rooted in the binary encoding proposed by Guo et al. [112] which eradicates the numerical meaning of the features completely.

For a quantitative analysis, we determine the distribution of the top 5 features from the LRP explanations for each class in the entire dataset and present the result in Table 4.1. It turns out that JavaScript appears in 88 % of the malicious documents, whereas only about 6 % of the benign samples make use of it. Similarly, `pdfid1_num` (number of numerical characters) is present almost exclusively in the benign class of the dataset. This makes these two features extremely discriminating for the dataset. From a security perspective, this is an unsatisfying result, as the neural network of Mimicus+ relies on a few indicators for classifying the documents. An attacker could potentially evade Mimicus+ by not using JavaScript or including numerical characters in the document.

Table 4.1: Top-5 features for the entire Mimicus+ dataset determined using LRP. The right columns show the total frequency in benign and malicious PDF documents, respectively.

Class	Feature	Benign	Malicious	Class	Feature	Benign	Malicious
Benign	count_font	98.4%	20.8%	Malicious	count_javascript	6.0%	88.0%
Benign	pdfid1_num	81.5%	2.8%	Malicious	count_js	5.2%	83.4%
Benign	title_num	68.6%	4.8%	Malicious	count_trailer	89.3%	97.7%
Benign	title_uc	68.6%	4.8%	Malicious	count_endobj	100.0%	99.6%
Benign	pos_eof_min	100.0%	93.4%	Malicious	count_action	16.4%	73.8%

Id	Prototype	Prototype
0	intent::android.action.SIG_STR	permission::SEND_SMS
1	call::system/bin/su	feature::android.hardware.telephony
2	call::getSubscriberId	real_permission::SEND_SMS
3	intent::action.BOOT_COMPLETED	permission::INTERNET
4	...	...
5	real_permission::READ_PHONE_STATE	intent::action.MAIN
5	real_permission::ACCESS_FINE_LOCATION	call::getSystemService
6	real_permission::ACCESS_WIFI_STATE	feature::android.hardware.touchscreen
7	intent::android.LAUNCHER	intent::category.LAUNCHER"

Figure 4.3: Prototypes of malicious Android applications from the Drebin+ dataset as presented by the first two steps of the Prof scheme.

ANDROID MALWARE DETECTION We continue our analysis by investigating the explanations for the neural networks for Android malware detection, namely Drebin+ and DAMD. We begin with the Drebin+ network that processes static string features in a high dimensional and sparsely populated bag-of-words feature space before turning to the DAMD model that leverages convolutions to process the de-compiled application code sequentially.

DREBIN+ ANALYSIS We present two protoypical malware examples traversed by the PROF schemes in Figure 4.3. The features speaking for the malicious classification can be linked to the functionality of the malware. For instance, the requested permission `SEND_SMS` or features related to accessing sensitive information, such as the call `getSubscriberId` receive consistently high scores in our investigation. These features are well in line with common malware for Android, such as the *FakeInstaller* family [179], which is known to obtain money from victims by secretly sending text messages (SMS) to premium services. We conclude that the MLP network employed in Drebin+ has captured combinations of indicative features directly related to the underlying malicious activities. Investigating the features speaking against a malicious classification, we notice the presence of the hardware feature `touchscreen`, the intent filter `LAUNCHER`, and the intent action `MAIN`. These features frequently also occur in the prototypes returned by the PROF scheme on benign applications. Investigating their distribution we find that they appear in both classes of the Drebin+ dataset and are thus not particularly descriptive for benignity. Note that the interpretation of features speaking for benign applications is challenging due to the broader scope and the difficulty in defining benignity. We therefore conclude that the three features together form an artifact in the dataset that provides an indicator for detecting benign applications.

DAMD ANALYSIS As a second Android malware detector, we consider applications from the DAMD dataset. In contrast to the static string features that constitute the datapoints of the Drebin+ dataset, analyzing raw Dalvik bytecode of thousands of instructions is not possible for a human anymore. Therefore, we guide our analysis of the dataset by inspecting malicious applications from three popular Android malware families whose behavior is well documented: GoldDream [140], DroidKungFu [139], and DroidDream [92]. These families exfiltrate sensitive data and run exploits to take full control of the device. In our analysis of the Dalvik bytecode, we benefit from the sparsity of the explanations from LRP and IG as explained in Section 3.3. Analyzing all relevant features becomes tractable with moderate effort using these methods and we are able to investigate the opcodes with the highest relevance in detail. In this setting, sparse explanations act like a compressed representation of the datapoints where only relevant parts of the applications must be investigated by the practitioner. Indeed, we observe that the relevant opcode sequences are linked to the malicious functionality of the three malware families.

In the last chapter we already had a closer look on a DAMD explanation in Table 3.2. This example depicts the opcode sequence, that is marked as highly important in all samples of the GoldDream family by LRP and IG. Using an Android decompiler we transfer the surrounding of this opcode sequence back to source code and find that it occurs in the `onReceive` method of the `com.GoldDream.zj.zjReceiver` class. In this function, the malware intercepts incoming SMS and phone calls and stores the information in local files before sending them to an external server. Similarly, we can interpret the explanations of the other two malware families, where functionality related to exploits and persistent installation is highlighted in the Dalvik opcode sequences. Members of the DroidDream family are known to root infected devices by running different exploits. If the attack has been successful, the malware installs itself as a service with the name `com.android.root.Setting` [92]. The top-ranked features appearing in the LRP explanation indeed lead to two methods of this very service, namely `com.android.root.Setting.getRawResource()` and `com.android.root.Setting.cpFile()`. Likewise, the highest ranked opcode sequence of the DroidKungFu family points to the class, in which the decryption routine for the root exploit is stored in.

The investigations above show that the CNN in the DAMD system has extracted discriminative patterns to detect malicious Android applications from the underlying opcode representation. This is a remarkable result considering the challenging dataset with sequences of thousands of tokens with variable length and demonstrates how explainable machine learning can assist the developer with the analysis of malware by pinpointing him to features and datapoints worth investigating.

4.2.2 Vulnerability Detection

Finally, we turn to the last security model from Chapter 3 to detect vulnerabilities in source code. In contrast to the datasets considered before, the features processed by VulDeePecker resemble sequences of lexical tokens embedded in a vector space which are strongly interconnected on a syntactical level. This setup does not allow to apply the Prof scheme directly, since the explanations are composed of a sequence of saliency scores $r_1, \ldots, r_d$ assigned to each token $s_1, \ldots, s_d$ of the code snippet. To embed each code gadget in a vector space, we leverage the embedding function e that maps tokens to vectors and represent an input sequence by the weighted sum of its embedded tokens, $\mathbf{r} = \sum_{i=1}^{d} r_i e(s_i)$.

Figure 4.4 shows a vulnerable code snippet that corresponds to a prototype according to the Prof scheme together with its explanation. We immediately observe the sequential nature of the dataset since tokens at the beginning aswell as at the end of the snippet receive relevance values with a concentration of high scores at the end of the first line. However, we also notice that it is very difficult for a human analyst to benefit from the highlighted tokens for two major reasons: Firstly, the pre-processing steps of VulDeePecker that replace variable names with generic ones and shorten the code snippet to 50 tokens can disconnect the vulnerable part of the code from the representation that is processed by the network. For example, in Figure 4.4, the size of the arrays is obfuscated due to the replacement of the numbers in the code gadget and thus makes it impossible to decide whether the `memmove` call overwrites memory or not. Similarly, in Figure 1.1, the highlighted `INT0` and `INT1` tokens as buffer sizes of 50 and 100 wide characters are ambiguous, since the neural network is not aware of the size relation due to the code shortening. Secondly, we observe a large amount of highlighted tokens corresponding to semicolons, brackets, and equality signs. These characters are common in C++ code due to the syntax but apparently can not indicate the presence of vulnerabilities. Therefore, we hypothesize that the VulDeePecker model overfitted the training dataset instead of learning an underlying concept of source code vulnerabilities. We will verify this claim through different experiments in the remainder of this subsection.

```
data = new char[10+1];
char source[10+1] = SRC_STRING;
memmove(data, source, (strlen(source) + 1) * sizeof(char));
```

```
VAR0 = new char [ INT0 + INT1 ] ;
char VAR1 [ INT0 + INT1 ] = VAR2 ;
memmove ( VAR0 , VAR1 , ( strlen ( VAR1 ) + INT1 ) * sizeof ( char ) ) ;
```

Figure 4.4: Top: Code snippet from the dataset. Bottom: Same code snippet after pre-processing steps of VulDeePecker explained by the LRP approach.

To see whether VulDeePecker relies on artifacts, we use the relevance values for the entire training set and extract the ten most important tokens for each code snippet. Afterwards we extract the tokens that occur most often in this top-10 selection and report the results in Table 4.2 in descending order of occurrence.

Table 4.2: The 10 most frequent tokens in the top 10 features of the entire VulDeePecker dataset.

Rank	Token	Occurrence	Rank	Token	Occurrence
1	`INT1`	70.8 %	6	`char`	38.8 %
2	`(`	61.1 %	7	`]`	32.1 %
3	`*`	47.2 %	8	`+`	31.1 %
4	`INT2`	45.7 %	9	`VAR0`	28.7 %
5	`INT0`	38.8 %	10	`,`	26.0 %

We can quantitatively confirm the observation that tokens such as '`(`', '`]`', and '`,`' are among the most important features throughout the training data although they occur frequently in code from both classes as part of function calls or array initialization. Secondly, there are many generic `INT*` values which frequently correspond to buffer sizes. From this we conclude that VulDeePecker is relying on combinations of artifacts in the dataset that are not connected to code vulnerabilities and thereby overfits the training dataset.

To further support this finding, we train a Logistic Regression classifier (see Example 2) on n-gram features of the code snippets as a simple baseline and also employ an ensemble of further standard models, such as random forests and AdaBoost classifiers, trained with the *AutoSklearn* library [90]. The classification performance and model size are reported in Table 4.3 and we find that the LR classifier with 3-grams yields a better performance than VulDeePecker with an 18× smaller model. This is interesting as overlapping but independent substrings (n-grams) are used, rather than the true sequential ordering of all tokens as for the RNN. Thus, it is likely that VulDeePecker is not exploiting relations in the sequence, but merely combines special tokens—an insight that could have been obtained by training a linear classifier. Furthermore, it is noteworthy that both baselines provide significantly higher true positive rates, although the AUC-ROC of all approaches only slightly differs.

Table 4.3: Performance of a Logistic Regression classifier, an ensemble of the AutoSklearn library and VulDeePecker on unseen data. The true-positive rate is determined at 2.9% false positives.

Model	# Parameters	AUC	TPR
Logistic Regression	6.6×10^4	0.982	0.961
AutoSklearn	8.5×10^5	0.982	0.894
VulDeePecker	1.2×10^6	0.984	0.818

Inspired by the analysis of the Damd network, we propose another application of explainable machine learning in the context of security, namely the vetting of malware *tags*. Coping with the sheer amount of new malware variants is a challenging and daunting task. Large security companies, for example, are required to process several hundred thousands of new samples per day [284]. This plethora of malicious code makes it hard to keep abreast of current malware development and update protection mechanisms in time. To alleviate this problem, it is common practice to flag incoming samples with short *tags* that label their origin, format, behavior, or family. These tags provide an indispensable tool for maintaining large collections and help focus investigations to particular files. VirusTotal, for example, features an extensive search engine for finding such tags in their database of malware samples [286].

Malware tags, however, greatly differ in purpose and quality. While a few tags are manually assigned based on careful reverse engineering, most tags are automatically derived from available information, such as file headers [298, 301], anti-virus labels [239, 240], clusterings [125, 138], and threat intelligence [96, 232]. Although these generated tags provide useful clues for analysis, they may become disconnected from the actual behavior of the malware. For example, tags derived by an imprecise YARA rule [15] might incorrectly flag functionality that is actually not present in the samples. When curating a large collection of malware, it is thus often unclear whether generated tags align with the behavior of the samples.

As a solution, we propose TagVet, a method for vetting tags in malware collections which builds on explainable machine learning and enables the practitioner to automatically link tags to behavioral patterns observed during dynamic analysis. To this end, we devise a CNN for predicting malware tags from monitored system calls and their arguments. Leveraging explanations of the network, TagVet uncovers its relation to specific system calls and links tags to behavioral patterns in retrospective, thereby explaining their semantic relations. Our approach helps to improve the quality of malware collections by exposing errors and inconsistencies in the tagging process even if the process itself is not available.

TagVet builds on the three analysis phases as presented in Figure 4.5. First, it executes tagged malware samples in a sandboxed environment and encodes the monitored behavior in a way suitable for analysis. Second, it trains a CNN as a surrogate model for the tagging process, such that tags can be predicted from monitored log files. Finally, LRP is used to link the tag predictions back to patterns in the monitored behavior. We will discuss each of the stages in detail in the remainder of this subsection and discuss insights we can obtain for a modern malware dataset afterwards.

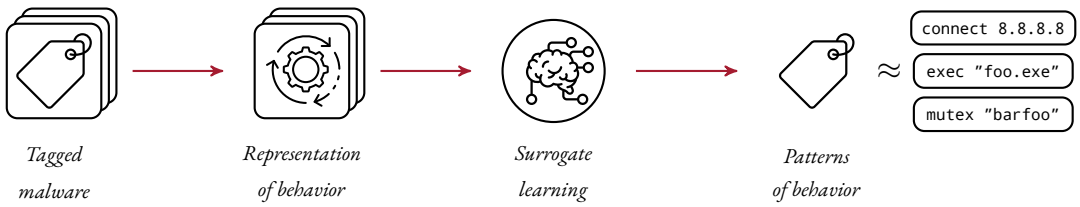


Figure 4.5: Schematic overview of the TagVet approach. First, tagged malware is collected. Second, behavior reports are collected from malware execution. Third, a surrogate model is trained to predict the tags from behavior reports. Fourth, explanations of the predicted tags are aggregated to behavioral patterns. Figure reproduced from Pirch et al. [210].

4.3.1 Behavior Monitoring and Representation

In the first stage of TagVet, malware samples are executed in a sandboxed environment to monitor their behavior dynamically. For our experiments, we employ *VMRay Analyzer*, a hypervisor-based sandbox for the Windows platform [287] that records all system calls to operating system libraries and the kernel. This dynamic analysis yields a behavior report for each executed sample that comprises lists (threads) of system calls with respective arguments. Our analysis hence operates on the boundary of the operating system and characterizes malware through its interaction with the host API.

Consolidation. As the sandbox reports contain very fine-granular data from the operating system state, we apply different consolidation steps before analyzing the behavior with a neural network. First, volatile call arguments, such as memory addresses and process identifiers, depend only on the system state and can safely be ignored. Second, we observe several function call arguments consisting of rare substrings, such as temporary file names. To consolidate these, we split each argument using appropriate delimiters ("\" for file and registry paths and "." for network addresses) and analyze the frequency of the resulting substrings. Substrings apprearing in less than 10 % of the reports are then unified by replacing them with a wildcard symbol "*".

Representation of behavior. To achieve a unified representation of the behavior described in the sandbox reports, we consider each report as a sequence of system calls and their arguments. For simplicity, we ignore the relation of threads in this representation, as the convolutional neural network used in TagVet can focus on local patterns in the data. Formally, we map a report to a sequence $x = (x_1, \dots x_l)$ of l tokens, where each token either corresponds to a system call or an argument. As the number of arguments varies between calls, we pad all arguments to a fixed number t using a special *pad* token. Consequently, if the report contains s system calls, the final sequence x has a length of $l = s \cdot (1 + t)$.

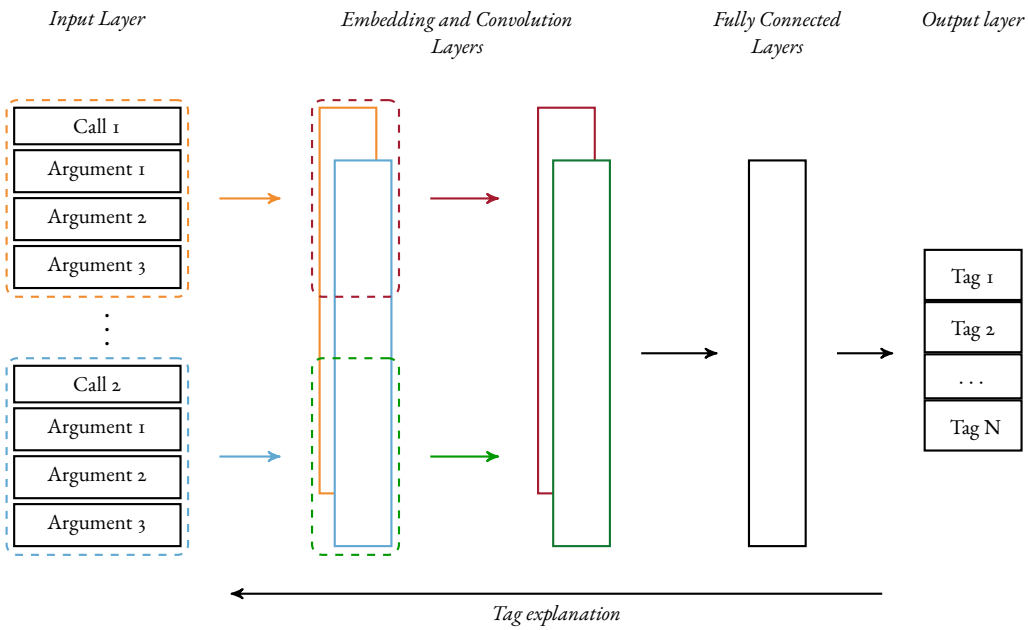


Figure 4.6: CNN for tag prediction: The first convolution captures system calls with their arguments, the second convolution summarizes multiple calls and the last layer returns tag probabilities. Figure reproduced from Pirch et al. [210].

4.3.2 Tag Learning and Prediction

For the next stage of TagVet, we require a machine-learning model that predicts tags for a given behavior. We refer to this model as a *surrogate model* as it mimics the original tagging process. While several approaches might be applicable for learning to predict tags, we employ the CNN from the Damd system due to the meaningful explanations that were extracted from it and due to the fact that it can be precisely tailored to the problem at hand.

Convolution of system calls. The network input consists of a padded sequence of tokens reflecting system calls and arguments. Hence, we compose the first convolutional layer of m_1 different filters of size $t + 1$ which are slid over the input, such that they process a complete system call with its argument at a time (Figure 4.6, left). To provide vector inputs for this convolution, we utilize an embedding layer which is also optimized during training. The second convolutional layer has m_2 filters and performs convolutions on the output of the first layer (Figure 4.6, middle). Hence, this layer infers dependencies between different system calls and captures broader patterns in malware behavior. Since the size of this convolution depends on the input length we employ a max-pooling layer that maps the output of the second convolutional layer to a vector of size m_2. This vector is finally fed to a fully-connected layer that returns probabilities for the tags of the input sample (Figure 4.6, right).

4.3.3 Generating Explanations

Once a surrogate model has been trained, we are able to apply techniques of explainable machine learning to interpret its prediction process and unveil behavior associated with the tags. As the input tokens correspond to system calls and arguments, this process enables us to pinpoint behavior relevant for specific tags as for the DAMD system. Following the insights of Chapter 3, we use LRP to compute the relevance scores in our approach and normalize the relevance scores to lie in the interval $[-1, 1]$.

As an example, Table 4.4 shows a simplified explanation generated for the behavior tag *„creates process with hidden window"* assigned by the VMRay Analyzer. The argument value `create_suspended` of the argument `creation_flags` that belongs to the system call `proc_create` obtains the highest relevance, as it is typically used to create a process for a background window. Other highlighted tokens, e.g. `sw_hide` or `show_window`, also fit perfectly to the tag.

Table 4.4: Explanation snippet for behavior tag "Creates process with hidden window" assigned from the VMRay analyzer.

Id	System Call and Arguments	Id	Argument Value
0	`proc_create`		
1	`symbol_name`	2	`createprocess`
3	`creation_flags`	4	`create_suspended`
5	`show_window`	6	`sw_hide`
7	`success`	8	`true`

Aggregating behavioral patterns The tokens with the strongest influence alone can be meaningless without the surrounding context. Imagine, for example, that the most relevant token for a tag corresponds to the argument value "true". As a remedy, we propose an aggregation scheme for the explanations of TAGVET, tailored to the context of program behavior. To represent the context of a token, we use a notation inspired by the Python programming language that builds on named arguments. For each relevant token, we identify the related system call and then compose an explanation describing this call with a named argument. As an example, for the snippet in Table 4.4, we write

```
proc_create(in:creation_flags="create_suspended"),
```

indicating that the relevant token `create_suspended` belongs to the system call `proc_create`. The prefix `in:` denotes an input argument whereas `out:` signifies a return value. To generate a behavioral pattern from such calls, we compute the surroundings of the 10 most relevant tokens in every sample and count their occurrences in the entire dataset. If multiple relevant

tokens belong to the same function call, we expand the aggregation accordingly. Then, we average the relevance values of the single aggregations and sort them by their occurrence in descending order to obtain behavioral patterns that encapsulate the entire execution log.

Evaluation We evaluate the effectivity of TagVet in a series of experiments with a dataset comprised of real-world malware. In particular, we explore how our method learns to predict tags and whether its explanations help to understand the underlying malware behavior. To this end, we first present a quantitative evaluation of our approach and then qualitatively discuss four case studies.

The dataset contains malware from the VirusShare repository [227] where we focus on samples that target the Windows platform. In particular, we retrieve a recent subset of 65,536 samples and extract all valid PE files, resulting in 6,598 malware samples. Each of these samples is then labeled by multiple virus scanners and we use AVClass [239] to determine their family labels. As we intend to simulate the vetting process in practice, we filter out very small families with less than 10 samples, resulting in 5,217 malware samples from 71 families. Finally, we execute each sample in the VMRay Analyzer sandbox [287] with simulated user traces and internet connectivity to monitor as much malicious behavior as possible.

Malware tags We consider three types of tags for our experiments: First, we use the family labels assigned by AVClass as *family tags*. Second, the VMRay Analyzer comes with 60 predefined threat indicators that are matched during monitoring and result in *sandbox tags*, such as *"creates process with hidden window"*. Third, we conduct a behavior-based clustering similar to Rabadi and Teo [216] to group similar reports into *clusters*, such that the difference between clusters is large. To employ clustering algorithms, it is necessary to define a distance metric between two behavior reports in our setup. Similar to Rieck et al. [222] we use tuples of the form (function call, argument) and count their occurrences in each report to obtain a vector representation of each log in a high dimensional vector space. Comparing multiple clustering algorithms on the data we find that the complete-linkage clustering [71] performs best in our experiments. To calibrate the algorithm in terms of the optimal number of clusters, we employ the Mean Silhouette Score [231] and the Adjusted Rand Index [129] as performance metrics which both assign a score between zero and one to a given partitioning of the data. Figure 4.7 shows both metrics for a different number of clusters. The optimal number of clusters can be found using the „Elbow method", i.e. searching for the point where small gains in performance are no longer worth separating the data further, and corresponds to 30 clusters in our case.

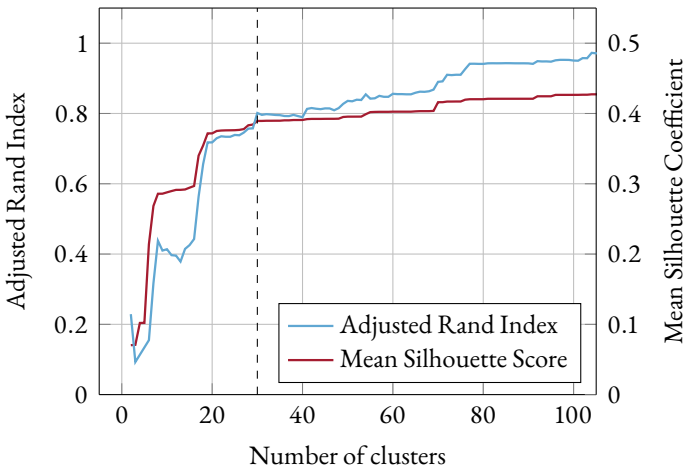


Figure 4.7: Mean Silhouette Coefficient and Adjusted Rand Index for different clusterings. The dashed line corresponds to 30 clusters that were chosen according to the Elbow method.

Setup of TagVet After consolidation, our dataset comprises 5,217 sequences with 4,241 unique tokens. We train the CNN model on this data using an embedding dimension of 128 and a fixed number of 21 system call arguments. The filter sizes of the two convolutional layers are 21 and 5, respectively, and we use 64 filters for each layer.

4.3.4 Quantitative Evaluation

We start our evaluation by investigating the prediction performance of TagVet and the quality of the generated explanations. For these experiments, we split our dataset into a training, validation, and test partition using 80 %, 10 %, and 10 % of the data, respectively. We train the CNN on the training data and use the validation partition to calibrate all model parameters. The final model is then applied to the (unseen) test data. This procedure is repeated five times, and the performance is averaged.

Prediction performance We measure the prediction performance of TagVet using three standard metrics introduced in Section 2.1, namely the *accuracy* the *area under the ROC curve*, which describes the relation between true-positive and false-positive predictions and the *area under precision-recall curve* that provides a view on the precision and recall of our approach. Thus far, we considered these metrics only for binary classification problems, however we face a multi-class prediction problem in our setup since one malware sample can have multiple tags. To this end, we calculate the metrics for every output class separately and compute a weighted average over the results. That is, tag classes that occur often have a proportionally higher weight in the final score than those appearing rarely.

Table 4.5: Prediction performance of TagVet. All performance metrics are averaged over the tag classes.

Tag Class	# Tags	Accuracy	AUC (ROC)	AUC (PR)
Sandbox	60	0.97 ± 0.002	0.99 ± 0.001	0.98 ± 0.002
Family	71	0.94 ± 0.007	0.96 ± 0.007	0.85 ± 0.015
Clustering	30	0.92 ± 0.019	0.99 ± 0.004	0.95 ± 0.013

Table 4.5 shows the prediction performance of TagVet for each tag class. We observe that the CNN can successfully predict the different tags for all types with high accuracy. The best results are achieved for the sandbox tags, likely because these are directly derived from the behavior monitored in the sandbox. The family and clustering tags, on the other hand, also yield a strong accuracy and justify the usage of explainable learning in our approach.

Explanation quality To evaluate the explanations generated by TagVet, we use the *descriptive accuracy (DA)* and the *descriptive sparsity (DS)* defined in Section 3.2. In Figure 4.8 we present the averaged scores for all tag classes, where we choose the *pad* token for the removal operation when computing DA. The DA score drops quickly for all tag classes, for example, removing the top 50 features reduces the accuracy by 16.7%, 22.4% and 14.4 % for the clustering, family and sandbox tags. Considering the large number of different tags and that features may only be important for a fraction of the classes, this result indicates a high quality of the explanations. Moreover, we find that the descent in the curve for the VTI rules is not as steep as for the other ones. We conjecture that this effect stems from the more complex tagging process, as one sample can be assigned multiple sandbox tags. For the DS, we observe that all curves have an extremely steep rise close to zero indicating that more than 90 % of the features are irrelevant and only a few important ones form the explanations.

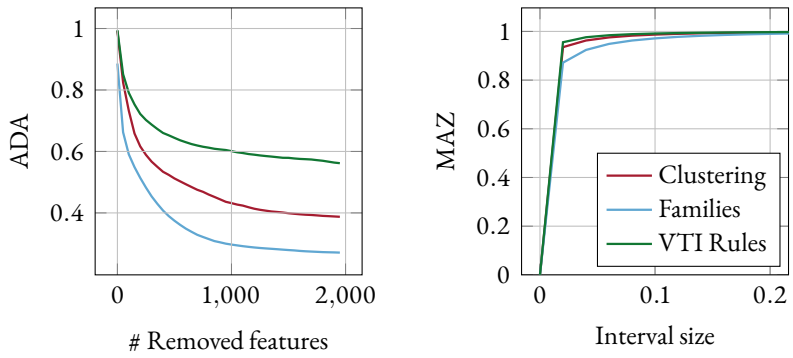


Figure 4.8: Average descriptive accuracy and Mass around Zero of TagVet for the different tag classes.

4.3.5 Qualitative Evaluation

The previous experiments demonstrate the quality of the learned model, yet the generated explanations need to also be sensible from the practitioner's perspective. We therefore perform four qualitative case studies to verify the semantic coherence of the extracted patterns with human expectations. Although it would be possible to apply the Prof scheme to select explanations, it is sufficient to consider the explanation of malware families whose behavior is known and behavior tags that correspond to actions we can link to the reports in order to evaluate the explanation quality in this setup.

Sandbox tags As the first case study, we inspect random samples of behavioral patterns for the 60 sandbox tags. We observe that all patterns align well with the names of the underlying threat indicators and a clear relation between behavior and the tag semantics. Table 4.6 shows an example for the tag *"attempts to connect to unavailable TCP servers"*. As expected, the tag is supported by the socket connection function (line 1) in combination with the unsuccessful invocation attribute (line 3), resulting in a concise summary of the threat indicator matched by the sandbox.

Table 4.6: Behavioral pattern for the sandbox tag "Attempts to connect to unavailable TCP servers".

Id	Tokens in context
1	`sck_connect(*)`
2	`sck_connect(in:post_symbol_name="connect")`
3	`sck_connect(out:success="false")`
4	`sck_connect(in:remote_port="*")`

Family tags Regarding the malware families, we observe much more specificity in the explanations. This makes sense because generic behavioral patterns such as accessing external IP addresses are occurring across different families and, therefore, are not useful for their explanation. In this case study, we examine the *Fareit* malware family, for which Table 4.7 shows the behavior explanation. We find that creating a window with `windprocparameter=0`as an argument appears among the top ten most relevant features in 79 % of the explanations of the family's samples. Also, 78 % of the *Fareit* samples sleep for exactly 25 s, whereas instances from the *Autoit* malware family, for example, typically sleep only for 750 ms. Judging by the steep drop in accuracy when removing these features (see Figure 4.8), we conclude that the rather detailed behavioral patterns are characteristic for specific malware families and are indeed suitable candidates for behavioral detection rules.

Table 4.7: Behavioral pattern for the malware family Fareit.

Id	Tokens in context
1	wnd_create(in:wndproc_parameter="0")
2	wnd_create(in:width="320")
3	sleep(in:milliseconds="25000")
4	sleep(out:milliseconds_text=""25000 milliseconds (25.000 seconds)")
5	wnd_create(in:class_name="t__304124810")

CLUSTER TAGS As the final category, we examine the explanations for tags that were generated by our clustering strategy. By a manual inspection, we observe that the overlap between clustering tags and family tags is high, which indicates a successful clustering procedure. However, we also find a few behavioral patterns that deviate from the respective families. Table 4.8 shows the output for cluster #15, which contains system calls for reading out environment variables and sending an HTTP request to "`ipv4bot.whatismyipaddress.com`". Interestingly, the presented tokens are not identical to the ones from the respective malware family but correspond to it semantically: The *Gandcrab* malware, that constitutes the largest number of samples in cluster #15, scans the user environment and determines its IP address when executed. Furthermore, we inspect some reports from that cluster to see whether the wildcard in the file system path conceals relevant information. This is not the case as the respective function call occurs but the wildcard only replaces an MD5 hash value in all of the inspected files.

Table 4.8: Behavioral pattern for the cluster #15.

Id	Tokens in context
1	env_get(in:symbol_name="getenvironmentvariable")
2	file_create(in:file_name_orig="c:\users \%USERNAME%\desktop*.exe")
3	str_len(in:string="pridur")
4	file_create(in:file_name="c:\users \%USERNAME%\desktop*.exe")
5	open_http_request(in:url="ipv4bot.whatismyipaddress.com/*")
6	open_connection(in:server= "ipv4bot.whatismyipaddress.com")

TAGGING INCONSISTENCIES In addition, we investigate how tag explanations enable human analysts to reason about the quality of tags. In this final case study, we examine tagging inconsistencies between the family and behavior tags. That is, finding cases in which for example one family tag is split across multiple behavioral clusters or vice-versa. Analyzing the explanations from the CNN operating on behavior reports can then give evidence about the discrepancies between behavioral tags and the ones generated using different analyses.

For instance, anti-virus products create family tags based on various sources of information, including file metadata and static analysis. Hence, we expect some family explanations to contain static artifacts in their explanations when malware cannot be discriminated from a behavioral perspective alone.

Through a manual analysis of multiple family explanations we find this to be the case for the *AutoIt* family. The origin of this family name refers to a Windows scripting language which suggests that the family tagging process is likely based on malware being written in that language. If this is the case, however, this would not be directly observable in behavioral reports since they do not contain information about the programming language. The following two findings support this hypothesis. First, the *Autoit* family explanation in Table 4.9 shows that the CNN relies on an artifact for correct classification. Concretely, the window creation function in line 5 contains the family name in one of its arguments. Scanning the explanation corpus for this value, we find that it is of high relevance when training the CNN to predict family tags but never occurs in explanations for behavior tags.

A second finding is that the 136 *AutoIt* samples are scattered across 15 behavioral clusters. The explanation for the cluster with the largest number of *AutoIt* samples (44/136) is shown in Table 4.10 and indeed contains some related behavior. According to the cluster explanation, loading the functions "VarSub", "VarMod" and "VarDiv" from a dynamically linked library accounts for three of the top five most relevant features and is present in more than 99 % of the explanations. These functions are used for basic arithmetic operations, are part of the Windows API (`oleauto.h`) and are internally used by the *AutoIt* language in version 3 which coincides with the artifact observed in Table 4.9. Yet, the functions are not exclusively available to this scripting engine and hence might be used by other malware families as well. Furthermore, the *AutoIt* samples comprise only 14.8 % of all samples in the analyzed cluster and explanations for other clusters with *AutoIt* samples contain no relatable information at all. We conclude from these weak behavioral indicators and the lack of explanation coherence between this family and the related clusters that TagVet is able to give evidence for reasoning about whether certain family tags are sensible from a behavioral perspective.

Table 4.9: Behavioral pattern for the malware family AutoIt.

Id	Tokens in context
1	`sys_sleep(in:milliseconds=750)`
2	`sys_sleep(out:milliseconds_text= "750 milliseconds (0.750 seconds)")`
3	`wnd_create(in:wnd_proc_parameter=0)`
4	`sys_sleep(in:post_symbol_name="settimer")`
5	`wnd_create(in:window_name="autoit v3")`

Table 4.10: Behavioral pattern for the cluster #28.

Id	Tokens in context
1	`wnd_create(in:wnd_proc_parameter=0)`
2	`mod_get_proc_address(in:module_name= "c:\windows\syswow64\user32.dll")`
3	`mod_get_proc_address(in:function="VarSub")`
4	`mod_get_proc_address(in:function="VarMod")`
5	`mod_get_proc_address(in:function="VarDiv")`

Our findings demonstrate the utility of our approach: Depending on the particular tags (sandbox, family, or clustering), only those features are identified that are relevant in the particular context. As a result, a cluster only partially overlapping with a malware family yields a different behavioral pattern and thus helps to understand what characteristics the tags reflect.

4.4 Related Work

In the following we review related work in the context of the enhancement of machine learning models in security by explanations and the dynamic analysis of malicious software.

Enhancing security models with explanations Explainable learning has been adapted to multiple learning models in security context in recent years. Among the first, Arp et al. [19] used the weights of a linear classifier to understand predictions for the detection of Android malwar. Han et al. [114] propose DeepAID to improve and explain anomaly detection systems based on neural networks. In a similar fashion, Wei et al. [299] introduce the xNIDS framework to respond to network intrusion detections based on saliency scores of the predictions. In the context of Android malware, explainable machine learning has been used to investigate the deteriorating performance of detection systems over time [59, 312], also called concept drift, and to develop general sanity checks for the explanations [89]. As a final example, Drichel et al. [78, 79] showed that detectors for *domain generation algorithms*, a specific approach used by malware authors to establish a connection to command and control servers, contain spurious correlations and therefore establish a false sense of security.

Malware Analysis and Tags Techniques for malware analysis can be roughly categorized into *static* and *dynamic* approaches. The former category comprises all techniques that inspect malicious code without executing it [e.g., 125, 198, 209, 246]. As static analysis suffers from inherent limitations [see 193], dynamic approaches need to complement it and expose functionality only observable at run-time [e.g., 75, 84, 152, 168, 216]. Several static

and dynamic approaches can be utilized to generate tags for malware automatically. For example, methods for clustering are a common and widely used tool for assigning cluster tags to malware files [e.g., 32, 138, 208, 223]. Similarly, information from file headers and metadata provides a valuable source for generating static tags [e.g., 298, 301].

Another branch of malware research has been concerned with methods for deriving family tags from anti-virus labels. Since these labels are notoriously inconsistent among virus scanners, these approaches apply different strategies for normalizing and consolidating the labels [e.g., 130, 131, 239, 240]. Recently, this strain of research has been further expanded with methods for deriving tags for general malware categories [240], tagging specific capabilities [215] and predicting functionality using threat intelligence [232]. Despite the breadth of this prior work, however, the retrospective explanation of tags has not been considered in malware research so far. Most tagging methods are black-box systems and remain opaque to the practitioner.

5

From Explanations to Unlearning

The previous chapters of this thesis demonstrated the remarkable prediction performance of neural networks in different setups. Analyzing the explanations, however, we also saw that they oftentimes rely on *artifacts* for their decisions, i.e. features that are unrelated to the learning task. This effect is unsatisfying as it allows adversaries to circumvent the classifier relatively easy, for example. Also, if the provider of a machine learning service decides to show customers not only the classification result but also an explanation to foster trust in the underlying system, the artifacts will be revealed to the users. This exposure can also affect sensitive information that is present in learning models nowadays. Carlini et al. [46] show, for example, that the Google text completion system contains credit card- and social security numbers that have been memorized from personal emails during training.

Removing such defects from a learning model is a challenging task that requires to partially revert the learning process. In the absence of specific methods, the only option is to retrain the model from scratch, which is costly and only possible if the original data is still available. As a remedy, methods for *machine unlearning* have been proposed recently [e.g. 38, 43]. These methods are capable of deleting data points in retrospection and thereby enable to mitigate privacy leaks and comply with removal requests from users under legislature like the General Data Protection Regulation (GDPR) or the California Consumer Privacy Act (CCPA).

In this chapter we take the viewpoint of a machine learning service provider who has to deal with the removal of privacy leaks manifested not only in isolated data points but over the entire dataset. When training on content of social media, sensitive data is often distributed across several data instances. The leaked home address of a celebrity, for example, may be shared in thousands of posts and its removal requires substantial changes to the model structure. Existing approaches for unlearning [e.g. 38, 197] are inefficient in these cases, as they operate on data points only: First, a runtime improvement can hardly be obtained over retraining when the changes are not isolated and larger parts of the data need to be corrected. Second, removing multiple data points reduces the fidelity of the corrected model and thus is not a viable option in practical scenarios. Consequently, unlearning should not be limited to removing data points, but allow corrections at different granularity of the training data, such as fixing leaks in features and labels individually.

To address these limitations, we propose the first method for unlearning features and labels from a learning model in this chapter. After formulating a mathematical framework for the unlearning problem in Section 5.1 we derive two closed-form update strategies in Section 5.2. As a theoretical concept for the success of unlearning we introduce the idea of *certified unlearning*, a strong mathematical guarantee for information removal building on certified data removal [110, 197] and differential privacy [51, 82] in Section 5.3. We demonstrate the applicability of the theoretical insights in Section 5.4 when removing features from a Logistic Regression classifier in a certified way. Although our unlearning guarantees cannot be realized for models with non-convex loss functions, such as deep neural networks, we demonstrate the efficacy and speed of our approach in case studies on unlearning unintended memorization from language models and label poisoning in computer vision. Finally, we review related work on machine unlearning approaches in Section 5.5.

Stochastic Analysis of Sharding To better understand the need for closed-form updates on model parameters, we examine current sharding strategies and investigate the circumstances under which they reach their limits. Bourtoule et al. [38] propose an unlearning method with the core idea to train separate models on distinct parts of training data. While the authors discuss limitations of their approach like performance degradation, we perform a stochastic analysis to find upper bounds for the number of affected data points at which sharding becomes as efficient as retraining.

In this context, we consider m data instances to unlearn which are uniformly distributed across s shards. Let $p(m)$ denote the probability that all shards contain at least one of these samples which leads to the worst-case scenario of having to retrain all shards. Since calculating

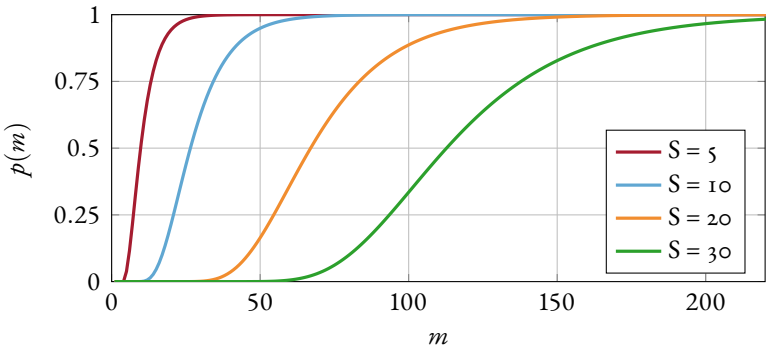


Figure 5.1: Probability of all shards being affected by unlearning for varying number of data points m and shards S.

$p(m)$ as stated above is difficult, we reformulate the task to solve an equivalent problem: We seek the probability $\hat{p}_k(m)$ that at most k shards remain unaffected by any sample. We set $k = s$ such that $\hat{p}_s(m)$ indicates the probability that any combination of $i \in \{1, \dots, s\}$ shards are unaffected. If this probability is zero there are no unaffected shards. Hence, this corresponds to the inverse of our target probability, $p(m) = 1 - \hat{p}_s(m)$.

To calculate $\hat{p}_s(m)$, we first determine the probability of exactly i shards to remain unaffected. In general, there are $(s-i)^m$ combinations to distribute m samples on the dataset excluding i shards. Since there are $\binom{s}{i}$ possible ways to select the i shards to be left out, the total number of combinations is given by $\binom{s}{i}(s-i)^m$. However, we cannot simply sum these terms up for different values of i since the unaffected shards in the combinations partly overlap. For example, a distribution of samples across 2 of 20 shards would be counted three times: One time for the combination of the two affected shards and one time for each single shard. To account for this, we apply the inclusion-exclusion principle and finally divide the adjusted term by the number of combinations including all shards to obtain

$$\hat{p}_s(m) = \frac{\sum_{i=1}^{s} (-1)^{i+1} \binom{s}{j} (s-i)^m}{s^m}.$$

Figure 5.1 shows the evolution of $p(m)$ and we see that the probability quickly reaches one even for low numbers of affected samples. Since the probability only depends on the number of shards and samples to unlearn and not on the size of the dataset, we can conclude that sharding is inefficient when there are many unlearning requests. This essentially motivates our approach using closed-form updates on model parameters.

To derive a mathematical formulation of our update strategy, we recall that the optimal parameters θ^* of a learning model can be found by minimizing the regularized empirical risk,

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\ L_{\mathbf{b}}(\theta; D) = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} \ell(z_i, \theta) + \lambda\Omega(\theta) + \mathbf{b}^T\theta, \tag{5.1}$$

where Ω a common regularizer as introduced in Chapter 2 and $\mathbf{b} \in \mathbb{R}^m$ is a random vector sampled from a given distribution. For conventional learning, this vector is set to zero and can be ignored. For realizing certified unlearning, however, it enables to add a small amount of noise to the optimization outcome, similar to differentially private learning [82, 83]. We introduce this technique later in Section 5.3 and omit the subscript in $L_{\mathbf{b}}$ for now.

Perturbing Data Points We begin the design of our approach by asking a simple question: How would the optimal model θ^* change, if only one data point z had been perturbed by some change δ? Replacing z by $\tilde{z} = (\mathbf{x} + \delta, y)$ leads to the new optimal model:

$$\theta^*_{z\to\tilde{z}} = \underset{\theta}{\operatorname{argmin}}\ L(\theta; D) + \ell(\tilde{z}, \theta) - \ell(z, \theta). \tag{5.2}$$

However, calculating the new model $\theta^*_{z\to\tilde{z}}$ exactly is expensive as it requires to solve the problem in Equation (5.1) again. Instead of replacing the data point z with $\tilde{z}$, we can also up-weight $\tilde{z}$ by a small value ε and down-weight z accordingly, resulting in the following adjusted problem formulation

$$\theta^*_{\varepsilon,z\to\tilde{z}} = \underset{\theta}{\operatorname{argmin}}\ L(\theta; D) + \varepsilon\ell(\tilde{z}, \theta) - \varepsilon\ell(z, \theta) \tag{5.3}$$

and we notice that Equations (5.2) and (5.3) are equivalent for $\varepsilon = 1$.

As a result, we do not need to explicitly remove a data point from the training data but can revert its *influence* on the learning model through a combination of up-weighting and down-weighting. It is easy to see that this approach is not restricted to a single point. We can define a set of points Z as well as their perturbed versions $\tilde{Z}$ to formulate the following optimization problem

$$\theta^*_{\varepsilon,Z\to\tilde{Z}} = \underset{\theta}{\operatorname{argmin}}\, L(\theta; D)\ + \varepsilon \sum_{\tilde{z}\in\tilde{Z}} \ell(\tilde{z}, \theta) - \varepsilon \sum_{z\in Z} \ell(z, \theta). \tag{5.4}$$

This generalization enables us to approximate changes on larger parts of the training data. Instead of solving the problem in Equation (5.4), however, we formulate the optimization as an update of the original model θ^*. That is, we seek a closed-form update $\Delta(Z, \tilde{Z})$ of the model parameters, such that

$$\theta^*_{\varepsilon, Z \to \tilde{Z}} \approx \theta^* + \Delta(Z, \tilde{Z}), \tag{5.5}$$

where $\Delta(Z, \tilde{Z}) \in \mathbb{R}^m$ affects only the weights necessary for unlearning. We show in Section 5.1 that this update step can be efficiently computed using first-order and second-order derivatives. If $\tilde{Z} = \emptyset$ in Equation (5.4), our approach also yields updates to remove *data points* similar to prior work [110, 150].

Unlearning Features and Labels Equipped with a general method for updating a learning model, we proceed to introduce our approach for unlearning features and labels. To this end, we expand our notion of perturbations and include changes to labels by defining

$$\tilde{z} = (\mathbf{x} + \delta_{\mathbf{x}}, y + \delta_y),$$

where $\delta_{\mathbf{x}}$ modifies the features of a data point and δ_y its label. By specifying different perturbations $\tilde{Z}$, we can now realize several unlearning tasks with closed-form updates.

Replacing features As the first type of unlearning task, we consider the task of correcting features in a learning model. This task is relevant if the content of some features violates the privacy of a user and needs to be replaced with alternative data. As an example, personal names, home addresses, or other sensitive information might need to be removed after a model has been trained on a corpus of emails. Similarly, in a credit scoring system, the race, gender or other biasing features might need to be replaced with neutral content. For a set of features $F \subset \{1, \ldots, d\}$ and their new values V, we define perturbations on the affected points Z by

$$\tilde{Z} = \left\{ (\mathbf{x}_{f=v}, y) : (\mathbf{x}, y) \in Z,\ (f, v) \in F \times V \right\},$$

where $\mathbf{x}_{f=v}$ denotes the datapoint $\mathbf{x}$ where dimension f is set to the value v. For example, a credit card number contained in the training data can be blinded by a random number sequence in this setting. It is noteworthy that the values V can be adapted individually for each data point in the definition above , so that fine-grained corrections become possible.

Replacing labels As the second type of unlearning task, we focus on correcting labels. This form of unlearning is necessary if the labels captured in a model contain unwanted or inappropriate information. For example, generative language models are oftentimes trained to predict the next word given a large text corpus. Therefore, the training text is used as input features (preceding tokens) *and* labels (target tokens) [105, 267]. Hence, defects can only be eliminated if labels can be unlearned as well. For the affected points Z and the set of new labels Y, we define the corresponding perturbation set for this unlearning task by

$$\tilde{Z} = \{(\mathbf{x}, y) \in Z_{\mathbf{x}} \times Y\},$$

where $Z_{\mathbf{x}}$ corresponds to the data points in Z without their original labels. The new labels Y can also be individually selected for each data point, as long as they come from the domain $\mathcal{Y}$, that is, $Y \subset \mathcal{Y}$. Note that the replaced labels and features can be easily combined in one set of perturbations $\tilde{Z}$, so that defects affecting both categories can be corrected in a single update. In Section 5.4, we demonstrate that this combination can be used to remove unintended memorization from generative language models with high efficiency.

Revoking features Based on appropriate definitions of Z and $\tilde{Z}$, our approach enables to replace the content of features and thus eliminate privacy leaks by overwriting sensitive data. In some scenarios, however, it might be necessary to even completely remove features from a learning model—a task that we denote as *revocation*. In contrast to the correction of features, this form of unlearning poses a unique challenge: The revocation of features reduces the input dimension of the model. While this adjustment can be easily carried out through retraining with adapted data, constructing a model update as in Equation (5.5) becomes tricky.

To address this problem, let us consider a model θ^* trained on a dataset D. If we remove some features F from this dataset and train the model again, we obtain a new optimal model $\theta^*{}_{-F}$ with reduced input dimension. By contrast, if we set the values of the features F to zero in the dataset and train again, we obtain an optimal model $\theta^*{}_{F=0}$ with the same input dimension as θ^*. These two models are equivalent for a large class of learning models, including several neural networks as the following lemma shows.

Lemma 1 For learning models processing inputs $\mathbf{x}$ using linear transformations of the form $\theta^T\mathbf{x}$, we have $\theta^*{}_{-F} \equiv \theta^*{}_{F=0}$.

Proof. It is easy to see that it is irrelevant for the dot product $\theta^T\mathbf{x}$ whether a dimension of $\mathbf{x}$

is missing or equals zero in the linear transformation

$$\theta_{-F}^T \mathbf{x} = \sum_{k:k\notin F} \theta_k x_k = \sum_k \theta_k 1\{k \notin F\} x_k = \theta_{F=0}^T \mathbf{x}.$$

As a result, the loss $\ell(z, \theta_{-F}) = \ell(z, \theta_{F=0})$ is identical for every data point z. Hence, $L(\theta; D)$ is also equal for both models and thus the same objective is minimized during learning, resulting in equal model parameters. □

Lemma 1 enables us to erase features from many learning models by first setting them to zero, calculating the parameter update, and then reducing the input dimension of the models accordingly. Concretely, to revoke the features F from a learning model, we first locate all data points where these features are non-zero with

$$Z = \big\{(x, y) \in D : x_f \neq 0, f \in F\big\}.$$

Then, we construct corresponding perturbations so that the features are set to zero, i.e.,

$$\tilde{Z} = \big\{(x_{f=0}, y) : (x, y) \in Z, f \in F\big\}.$$

Finally, we adapt the input dimension by removing the affected inputs of the learning models, such as the corresponding neurons in the input layer of a neural network.

5.2 Update Steps for Unlearning

Our approach rests on changing the influence of training data in a closed-form update. In the following, we derive two strategies for calculating this update. The first strategy builds on the gradient of the loss function and thus can be applied to any model with differentiable loss. The second strategy incorporates second-order derivatives which limits the application to loss functions with an invertible Hessian matrix.

First-Order Update Recall that we aim to find an update $\Delta(Z, \tilde{Z})$ that we can add to our model θ^* for unlearning. If the loss ℓ is differentiable, we can compute an optimal *first-order update* as follows

$$\Delta(Z, \tilde{Z}) = -\tau\Big(\sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta \ell(\tilde{z}, \theta^*) - \sum_{z\in Z} \nabla_\theta \ell(z, \theta^*)\Big) \tag{5.6}$$

where τ is a small constant that we refer to as *unlearning rate*.

To derive the first-order update, let us recall the optimization problem for the corrected learning model from Section 5.1:

$$\begin{aligned}\theta^*_{\varepsilon,z\to\tilde{z}} &= \operatorname*{argmin}_{\theta} L(\theta; D) + \varepsilon\ell(\tilde{z}, \theta) - \varepsilon\ell(z, \theta) \\ &= \operatorname*{argmin}_{\theta} L_\varepsilon(\theta; D),\end{aligned} \tag{5.7}$$

where $L_\varepsilon(\theta; D)$ is a combined loss function containing our update and the regularized loss $L(\theta; D)$. If ε is small and ℓ is differentiable with respect to θ, we can approximate $L_\varepsilon(\theta; D)$ using a first-order Taylor series at θ^* by

$$\begin{aligned}L_\varepsilon(\theta^*_{\varepsilon,z\to\tilde{z}}; D) \approx{}& L(\theta^*; D) + \varepsilon\big(\ell(\tilde{z}, \theta^*) - \ell(z, \theta^*)\big) \\ &+\Delta(Z, \tilde{Z}) \cdot \Big(\nabla_\theta L(\theta^*; D) + \varepsilon\big(\nabla_\theta\ell(\tilde{z}, \theta^*) - \nabla_\theta\ell(z, \theta^*)\big)\Big).\end{aligned}$$

Since $\theta^*_{\varepsilon,z\to\tilde{z}}$ is a minimum of $L_\varepsilon(\cdot; D)$, we have $L_\varepsilon(\theta^*_{\varepsilon,z\to\tilde{z}}; D) < L_\varepsilon(\theta^*; D)$. Incorporating this assumption in the Taylor series and using the condition that $\nabla_\theta L(\theta^*; D) = 0$, we now arrive at

$$\varepsilon\Delta(Z, \tilde{Z}) \cdot \Big(\nabla_\theta\ell(\tilde{z}, \theta^*) - \nabla_\theta\ell(z, \theta^*)\Big) < 0.$$

As we have $\varepsilon > 0$, we can continue to focus on the dot product of the equation. For two vectors $\mathbf{u}$, $\mathbf{v}$ the dot product can be written as

$$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\| \cos(\mathbf{u}, \mathbf{v}),$$

where $\cos(\mathbf{u}, \mathbf{v})$ is the cosine between the vectors $\mathbf{u}$ and $\mathbf{v}$. The minimum of the cosine is -1 which is achieved when $\mathbf{u} = -\mathbf{v}$, hence we have

$$\Delta(Z, \tilde{Z}) = -\big(\nabla_\theta\ell(\tilde{z}, \theta^*) - \nabla_\theta\ell(z, \theta^*)\big)$$

This result indicates that $\nabla_\theta\ell(\tilde{z}, \theta^*) - \nabla_\theta\ell(z, \theta^*)$ is the optimal direction to move starting from θ^*. The actual step size, however, is unknown and must be adjusted by a small constant τ yielding the update step defined above,

$$\theta^*_{\varepsilon,z\to\tilde{z}} = \theta^* - \tau\big(\nabla_\theta\ell(\tilde{z}, \theta^*) - \nabla_\theta\ell(z, \theta^*)\big).$$

Due to the linearity of the gradient, this derivation can be equally performed when multiple points are affected by the unlearning step.

Intuitively, this update shifts the model parameters from $\sum_{z \in Z} \nabla \ell(z, \theta^*)$ to $\sum_{\tilde{z} \in \tilde{Z}} \nabla \ell(\tilde{z}, \theta^*)$ where the size of the update step is determined by the rate τ. This update strategy is thus similar to a gradient descent update step, however, it differs in that it moves the model to the *difference* in gradient between the original and perturbed data, which minimizes the loss on $\tilde{z}$ and at the same time removes the information contained in z.

The first-order update is a simple and yet effective strategy since the gradients of the loss function ℓ can be computed in $\mathcal{O}(m)$ [207]. However, it involves a parameter τ that controls the magnitude of the unlearning. In Section 5.4 we will see that the unlearning rate can be calibrated using another metric that measures the sucess of unlearning.

Second-Order Update If we assume that ℓ is twice differentiable and strictly convex, we can build on the concept of influence functions [64, 113] to define the Second Order update, given by

$$\Delta(Z, \tilde{Z}) = -H_{\theta^*}^{-1} \left(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*) \right),$$

where $H_{\theta^*}^{-1}$ is the inverse Hessian of the loss at θ^* as introduced in Example 5. To derive this update, we recall that the optimality conditions of Equation (5.7) state that

$$0 = \nabla L({\theta^*}_{\varepsilon, z \to \tilde{z}}; D) + \varepsilon \nabla \ell(\tilde{z}, {\theta^*}_{\varepsilon, z \to \tilde{z}}) - \varepsilon \nabla \ell(z, {\theta^*}_{\varepsilon, z \to \tilde{z}}).$$

If ε is sufficiently small, we can again apply a Taylor approximation series to obtain

$$\begin{aligned} 0 \approx\ & \nabla L(\theta^*, D) + \varepsilon \nabla_\theta \ell(\tilde{z}, \theta^*) - \varepsilon \nabla_\theta \ell(z, \theta^*) \\ & + ({\theta^*}_{\varepsilon, z \to \tilde{z}} - \theta^*) \cdot \nabla^2 L(\theta^*, D) \\ & + \varepsilon \nabla^2 \ell(\tilde{z}, \theta^*) - \varepsilon \nabla^2 \ell(z, \theta^*). \end{aligned}$$

Since we know that $\nabla L(\theta^*; D) = 0$ by the optimality of θ^*, we can rearrange this solution using the Hessian of the loss function, so that we get

$${\theta^*}_{\varepsilon, z \to \tilde{z}} - \theta^* = -H_{\theta^*}^{-1} \left(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*) \right) \varepsilon, \tag{5.8}$$

where we additionally drop all terms in $\mathcal{O}(\varepsilon)$. Diving by ε we can expres the solution in terms of the influence of ε on the model and arrive at

$$\left. \frac{\partial {\theta^*}_{\varepsilon, z \to \tilde{z}}}{\partial \varepsilon} \right|_{\varepsilon = 0} = -H_{\theta^*}^{-1} \left(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*) \right).$$

Thus, a linear approximation around θ^* with $\varepsilon = 1$ yields the Second Order update

$$\theta^*_{z \to \tilde{z}} \approx \theta^* - H_{\theta^*}^{-1} \left(\nabla_\theta \ell(\tilde{z}, \theta^*) - \nabla_\theta \ell(z, \theta^*) \right).$$

Since all operations are linear, we can extend this update to multiple data points and finally obtain the *second-order update* for our approach:

$$\Delta(Z, \tilde{Z}) = -H_{\theta^*}^{-1} \Big(\sum_{\tilde{z} \in \tilde{Z}} \nabla_\theta \ell(\tilde{z}, \theta^*) - \sum_{z \in Z} \nabla_\theta \ell(z, \theta^*) \Big). \tag{5.9}$$

Note that this update does not require parameter calibrations, since the weighting of the changes is directly derived from the inverse Hessian of the loss function. The second-order update is the preferred strategy for unlearning on models with a strongly convex and twice differentiable loss function that guarantee the existence of $H_{\theta^*}^{-1}$. Although, the update step in Equation (5.9) can be easily calculated with common machine learning frameworks, the computation involves the inverse Hessian matrix, which can be difficult to construct for large learning models. In the following, we discuss some strategies that still allow to apply Second Order updates for large learning models.

Calculating the inverse Hessian Given the model parameters $\theta \in \mathbb{R}^m$, forming and inverting the Hessian requires $\mathcal{O}(nm^2 + m^3)$ time and $\mathcal{O}(m^2)$ space [150]. For models with a small number of parameters, the matrix can be pre-computed and explicitly stored, such that each subsequent request for unlearning only involves a simple matrix-vector multiplication.

To apply second-order updates for large learning models, like neural networks, we have to avoid storing the Hessian matrix H explicitly and still be able to compute $H^{-1}\mathbf{v}$. To this end, we rely on the scheme proposed by Agarwal et al. [11] for computing expressions of the form $H^{-1}\mathbf{v}$. This scheme requires to only calculate $H\mathbf{v}$ and avoids storing H^{-1}. The resulting *Hessian-Vector-Products* (HVPs) allow us to calculate $H\mathbf{v}$ efficiently by making use of the linearity of the gradient

$$H\mathbf{v} = \nabla_\theta^2 L(\theta^*; D)\mathbf{v} = \nabla_\theta \big(\nabla_\theta L(\theta^*; D)\mathbf{v} \big).$$

If we denote first j terms of the Taylor expansion of H^{-1} by $H_j^{-1} = \sum_{i=0}^{j} (I - H)^i$, we can recursively define the approximation $H_j^{-1} = I + (I - H)H_{j-1}^{-1}$. Now, if $|\lambda_i| < 1$ for all eigenvalues λ_i of H, we have $H_j^{-1} \to H^{-1}$ for $j \to \infty$. To ensure this convergence, we add a small damping term λ to the diagonal of H and scale down the loss function by

Algorithm 1: Second Order parameter update

Input: model θ^*, loss functions L and ℓ, order o, unlearning rate τ, batch-size B, iterations m, damping d, scale s, repetitions r
Output: Parameter update $\Delta(Z, \tilde{Z})$
Data: $D, D', Z, \tilde{Z}$

$\mathbf{g}_1 = \sum_{\tilde{z}\in\tilde{Z}} \nabla_\theta \ell(\tilde{z}, \theta^*)$, $\mathbf{g}_2 = \sum_{z\in Z} \nabla_\theta \ell(z, \theta^*)$
$\mathbf{v} = \mathbf{g}_1 - \mathbf{g}_2$
if $o == 1$ **then**
    $\Delta = -\tau\mathbf{v}$
else
    $\Delta = 0$
    for i=1:r **do**
        $\theta_{\text{new}} = 0$
        for j=1:m **do**
            batch = sample(D', size=B)
            hvp $= \nabla_\theta\left(\mathbf{v}^T \nabla_\theta L(\text{batch}, \theta^*)\right)$
            $\theta_{\text{new}} = \mathbf{v} + (1-d)\theta_{\text{new}} - \text{hvp}/s$
        $\Delta = \Delta + \theta_{\text{new}}/r$
return Δ

some constant which does not change the optimal parameters θ^*. We can then formulate the following algorithm for computing an approximation of $H^{-1}\mathbf{v}$: Given data points $z_1, \ldots, z_t$ sampled from D, we define the iterative updates

$$\tilde{H}_0^{-1}\mathbf{v} = \mathbf{v},$$
$$\tilde{H}_j^{-1}\mathbf{v} = \mathbf{v} + \left(I - \nabla_\theta^2 L(z_i, \theta^*)\right)\tilde{H}_{j-1}^{-1}\mathbf{v}.$$

In each update step, H is estimated using a single data point and we can use HVPs to evaluate the appearing product term $\nabla_\theta^2 L(z_i, \theta^*)\tilde{H}_{j-1}^{-1}\mathbf{v}$ efficiently in $\mathcal{O}(m)$ as demonstrated by Pearlmutter [207]. Averaging batches of data points further speeds up the approximation. Choosing t large enough so that the updates converge and averaging r runs to reduce the variance of the results, we obtain $\tilde{H}_t^{-1}\mathbf{v}$ as our final estimate of $H^{-1}\mathbf{v}$ in $\mathcal{O}(rtm)$ of time.

The entire approximation procedure is clarified as pseudo-code in Algorithm 1 and we will see in Section 5.4 that we can compute an unlearning update for a recurrent neural network with 3.3 million parameters in less than 30 seconds.

The machine learning service provider aims at reliably removing privacy issues and sensitive data from the deployed learning models. The process should ideally build on theoretical guarantees to enable *certified unlearning*, where the corrected model is stochastically indistinguishable from the one created by retraining. In the following, we derive conditions under which the second-order updates of our approach provide certified unlearning. To this end, we build on the concepts of *differential privacy* [82] and *certified data removal* [110], and adapt them to the unlearning task.

Let $\mathcal{A}$ be a learning algorithm that outputs a model $\theta \in \Theta$ after training on a dataset D, that is, $\mathcal{A} : D \to \Theta$. Randomness included in $\mathcal{A}$ induces a probability distribution over the output models in Θ. Moreover, we consider an unlearning method $\mathcal{U}$ that maps a model θ to a corrected model $\theta_{\mathcal{U}} = \mathcal{U}(\theta, D, D')$ where D' denotes the dataset containing the perturbations $\tilde{Z}$. The concept of *ε-certified unlearning* can then be defined as follows

Definition 7 Given some $\varepsilon > 0$ and a learning algorithm $\mathcal{A}$, an unlearning method $\mathcal{U}$ is ε-certified if

$$e^{-\varepsilon} \leq \frac{P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big)}{P\big(\mathcal{A}(D') \in \mathcal{T}\big)} \leq e^{\varepsilon}$$

holds for all $\mathcal{T} \subset \Theta$, D, and D'.

This definition ensures that the log-likelihood to obtain a model using the unlearning method $\mathcal{U}$ and training a new model on D' from scratch deviates at most by ε. Following the work of Guo et al. [110], we define *(ε, δ)-certified unlearning*, a relaxed version of ε-certified unlearning, as follows.

Definition 8 Under the assumptions of Definition 7, an unlearning method $\mathcal{U}$ is (ε, δ)-certified if

$$P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big) \leq e^{\varepsilon} P\big(\mathcal{A}(D') \in \mathcal{T}\big) + \delta$$

and

$$P\big(\mathcal{A}(D') \in \mathcal{T}\big) \leq e^{\varepsilon} P\Big(\mathcal{U}\big(\mathcal{A}(D), D, D'\big) \in \mathcal{T}\Big) + \delta$$

hold for all $\mathcal{T} \subset \Theta$, D, and D'.

This definition allows the unlearning method $\mathcal{U}$ to slightly violate the conditions from Definition 8 by a constant δ. Using the definitions above, it becomes possible to derive conditions under which our model updates realizes certified unlearning.

Relation to Differential Privacy Our definition of certified unlearning shares interesting similarities with the concept of differential privacy [82] that we highlight in the following. First, let us recall the definition of differential privacy for a learning algorithm $\mathcal{A}$:

Definition 9 Given some $\varepsilon > 0$, a learning algorithm $\mathcal{A}$ is said to be ε-differentially private (ε-DP) if

$$e^{-\varepsilon} \leq \frac{P\big(\mathcal{A}(D) \in \mathcal{T}\big)}{P\big(\mathcal{A}(D') \in \mathcal{T}\big)} \leq e^{\varepsilon}$$

holds for all $\mathcal{T} \subset \Theta$ and datasets D, D' that differ in one sample. While the difference in one sample is oftentimes defined such that $|D'| = |D| - 1$ we use a variation where $|D| = |D'|$ but one sample has been replaced, a concept denoted as "bounded differential privacy" in literature [73, 144].

We see that differential privacy is a *sufficient* condition for certified unlearning by simply setting the unlearning method $\mathcal{U}$ to the identity function in Definition 7. That is, if we cannot distinguish whether $\mathcal{A}$ was trained with a point z or its modification z_δ, we do not need to worry about unlearning z later. Consequently, certified unlearning can be obtained through DP, yet the learning model's performance may suffer when enforcing strong privacy guarantees [see 2, 51]. Later, we will see that the models obtained using our update strategies are much closer to $\mathcal{A}(D')$ in terms of performance compared to DP alone. In this light, certified unlearning can be seen as a compromise between the high privacy guarantees of DP and the optimal performance achievable by costly re-training.

A practical way to achieve DP for an optimization algorithm $\mathcal{A}$, first proposed by Chaudhuri and Monteleoni [50], uses the noise term $\mathbf{b}$ in Equation (5.1) to add noise to the optimal parameters. Concretely, it can be shown that $\mathcal{A}$ is differential private when $\mathbf{b}$ is distributed according to the density $\nu(\mathbf{b})$ given by

$$\nu(\mathbf{b}) = \frac{1}{\alpha} e^{-\beta\|\mathbf{b}\|}, \tag{5.10}$$

where α is a normalization constant and β depends on the privacy budget ε. Due to the similarity of certified unlearning and DP we can employ similar approaches to guarantee certified unlearning for our update strategies.

Certified Unlearning of Features and Labels To obtain certified unlearning guarantees for our approach, we make two basic assumptions on the employed learning algorithm: First, we assume that the loss function ℓ is twice differentiable and strictly convex, so that H^{-1} always exists and the second-order update is applicable. Second, we consider an

L_2 regularization in the optimization problem Equation (5.1), that is, the regularizer $\Omega(\theta)$ is given by $\frac{1}{2}\|\theta\|_2^2$. Both assumptions are satisfied by a wide range of learning models, for example the logistic regression from Example 2 but do not hold for neural networks due to their non-convex nature.

A helpful tool for analyzing the task of unlearning is the *gradient residual* $\nabla L(\theta; D')$ for a given model θ and a corrected dataset D'. For strongly convex loss functions, the gradient residual is zero *if and only if* θ equals $\mathcal{A}(D')$ since in this case the optimum is unique. Therefore, the norm of the gradient residual $\|\nabla L(\theta; D')\|_2$ reflects the distance of a model θ from the one obtained by retraining on the corrected dataset D'. The gradient residual $\mathbf{r}$ of $L_{\mathbf{b}}$ is given by

$$\mathbf{r} = \nabla L_{\mathbf{b}}(\theta; D') = \sum_{z \in D'}^{n} \nabla \ell(z, \theta) + \lambda\theta + \mathbf{b}$$

and differs from the gradient residual of L only by $\mathbf{b}$. Due to the randomness of $\mathbf{b}$ we cannot bound the gradient residual of $L_{\mathbf{b}}$ absolutely, however if we have an upper bound for L we can use a confidence interval of the distribution of $\mathbf{b}$, for example, to bound $L_{\mathbf{b}}$ statistically. If we can further bound the difference between the outcomes of retraining and unlearning, we can derive a probability distribution for $\mathbf{b}$ such that certified unlearning can be realized, similar to Chaudhuri and Monteleoni [50] and Guo et al. [110]. The detailed proofs for the following theorems are presented in Appendix A.1, for a deeper understanding for the concepts from convex optimization that are applied in the proofs, we recommend the book of Boyd and Vandenberghe [39].

Theorem 1 Assume that $\|\mathbf{x}_i\|_2 \leq 1$ for all data points and the gradient $\nabla \ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let $\tilde{Z}$ change the features $j, \ldots, j+F$ by magnitudes at most $m_j, \ldots, m_{j+F}$. If $M = \sum_{j=1}^{F} m_j$ the following upper bounds hold:

1. For the first-order update of our approach, we have

$$\left\|\nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right)\right\|_2 \leq (1 + \tau\gamma n)\gamma_z M|Z|$$

2. If the Hessian $H_{\theta^*}(z, \theta)$ is γ'-Lipschitz with respect to θ, we have

$$\left\|\nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right)\right\|_2 \leq \gamma'\left(\frac{M\gamma_z}{\lambda}\right)^2 n|Z|^2$$

for the second-order update of our approach.

Theorem 1 states an upper bound for the gradient residual produced by our update strategies and thereby a first quantification of their quality. In general, we observe that the bounds for the first- and second order updates are in $\mathcal{O}(n|Z|)$ and $\mathcal{O}(n|Z|^2)$ respectively. In order to obtain a small gradient residual norm the unlearning rate should be small, ideally in the order of $1/n\gamma$ and the Lipschitz constants should be as small as possible aswell. Since $\|\mathbf{x}_i\|_2 \leq 1$ we have $m_j \ll 1$ if d is large and thus M acts as an additional damping factor for both updates when changing or revoking features. For the second Order update, a stronger L^2 regularization with λ shrinks the gradient residual since the Newton step is exact for quadratic loss functions.

As a next step, we bound the difference between retraining, i.e. applying $\mathcal{A}$ to D', and our unlearning strategies $\mathcal{U}$. If $\mathcal{A}(D')$ is an exact minimizer of $L_{\mathbf{b}}$ on D' with density $f_{\mathcal{A}}$ and $\mathcal{U}(\mathcal{A}(D), D, D')$ is an approximated minimum with density $f_{\mathcal{U}}$, then Guo et al. [110] show that the difference between $f_{\mathcal{A}}$ and $f_{\mathcal{U}}$ for the model θ produced by $\mathcal{U}$ can be bounded using the following theorem.

Theorem 2 (Guo et al. [110]) Let $\mathcal{U}$ be an unlearning method with a gradient residual r with $\|r\|_2 \leq \varepsilon'$. If the vector $\mathbf{b}$ is drawn from a probability distribution with density p satisfying that for any $\mathbf{b}_1, \mathbf{b}_2 \in \mathbb{R}^d$ there exists an $\varepsilon > 0$ such that $\|\mathbf{b}_1 - \mathbf{b}_2\| \leq \varepsilon'$ implies $e^{-\varepsilon} \leq \frac{p(\mathbf{b}_1)}{p(\mathbf{b}_2)} \leq e^{\varepsilon}$ then

$$e^{-\varepsilon} \leq \frac{f_{\mathcal{U}}(\theta)}{f_{\mathcal{A}}(\theta)} \leq e^{\varepsilon}$$

for any θ produced by the unlearning method $\mathcal{U}$.

Theorem 2 allows us to proove certified unlearning by applying the bounds for the gradient residual from Theorem 1 and finding a suitable density function. As it turns out, the density $\nu(\mathbf{b})$ stated above in Equation (5.10) fulfils the properties required for Theorem 2.

Theorem 3 Let $\mathcal{A}$ be the learning algorithm that returns the unique minimum of $L_{\mathbf{b}}(\theta; D')$ and let $\mathcal{U}$ be an unlearning method that produces a model $\theta_{\mathcal{U}}$. If $\|\nabla L(\theta_{\mathcal{U}}; D')\|_2 \leq \varepsilon'$ for some $\varepsilon' > 0$ we have the following guarantees.

1. If $\mathbf{b}$ is drawn from a distribution with density $p(\mathbf{b}) = e^{-\frac{\varepsilon}{\varepsilon'}\|\mathbf{b}\|_2}$ then $\mathcal{U}$ performs ε-certified unlearning for $\mathcal{A}$.

2. If $p \sim \mathcal{N}(0, c\varepsilon'/\varepsilon)^d$ for some $c > 0$ then $\mathcal{U}$ performs (ε, δ)-certified unlearning for $\mathcal{A}$ with $\delta = 1.5e^{-c^2/2}$.

Theorem 3 finalizes our theoretical insights regarding certified unlearning guarantees. As a last modification, we adjust the update strategies to distribute the changes in $\tilde{Z}$ to multiple update steps.

Multiple Unlearning Steps So far, we have considered unlearning as a one-shot strategy. That is, all changes are incorporated in $\tilde{Z}$ before performing an update. However, Theorem 1 shows that the error of the updates rises linearly with the number of affected points and the size of the total perturbation. Instead of performing a single update, it thus becomes possible to split $\tilde{Z}$ into T subsets and conduct T consecutive updates. In terms of run-time it is easy to see that the total number of gradient computations remains the same for the first-order update. For the second-order strategy, however, multiple updates require calculating the inverse Hessian for each intermediate step, which increases the computational effort.

In terms of unlearning certifications, it is possible to extend Theorem 1 and show that the gradient residual bound rises linearly after T update steps, which allows to compute certification relatively easy.

Lemma 2 If the gradient residual of a single update step is bounded by C then the gradient residual after T consecutive update steps is bounded by TC.

5.4 Applications

Equipped with a solid theoretical foundation of our update steps we test their performance in practice. We firstly evaluate the removal of sensitive features in a setup where certified unlearning is possible before moving to neural networks for image- and language processing where assumptions like convexity and Lipschitz-continuity do not hold anymore.

5.4.1 Unlearning Sensitive Features

In our first unlearning scenario, we focus on the removal of sensitive features from learning models with strongly convex loss functions. In particular, we consider the logistic regression classifier introduced in Example 2 on real-world datasets. We employ datasets for spam filtering [183], Android malware detection [19], diabetes forecasting [81] and prediction of income based on census data [80]. An overview of the datasets and their basic statistics is given in Table 5.1.

Table 5.1: Overview of the four datasets used throughout the experiments for unlearning sensitive features.

	Spam	**Drebin+**	**Adult**	**Diabetis**
Data points	33,716	49,226	48,842	768
Features	4,902	2,081	81	8

We divide each dataset into a training and test set with a ratio of 80 % and 20 %, respectively. To create a feature space for learning, we use the numerical features of the Adult and Diabetis dataset as is, while for the Spam and Drebin+ datasets we extract *bag-of-words features*. That is, we represent each email (malware) by the words (capabilities) it contains and construct corresponding feature vectors [see 19, 183]. The Drebin+ dataset is smaller compared to Chapter 3 since we use only features that appear in at least 0.1% of all datapoints and remove duplicates that appear due to this procedure. Finally, we train logistic regression models for all four datasets.

Sensitive features To gain insights on the features of the learning models, we can employ the Gradients explanation method due to its versatility. While we observe several features with high weights in the models, we also discover sensitive information. For instance, we identify several first and last names as well as postal zip codes in the features of the Spam dataset. Similarly, we note that features likes martial status, sex and race in the Adult dataset reflect problematic information potentially discriminating individuals. Although these features may not appear to be a significant privacy violation at first glance, they lead to multiple problems: First, if the models are shared or made public, these features may reveal the identity of individuals in the training data. Second, features like names probably represent artifacts and thus bias spam filtering for specific individuals, for example, those having similar names or postal zip codes. Third, features like sex and race may introduce a bias into the income prediction that discriminates certain individuals.

Since two of the datasets are high-dimensional and contain sparse data (Spam and Drebin+), while the other two are low-dimensional with dense feature vectors (Adult and Diabetis) we introduce two setups for our unlearning experiments based on the insights obtained from the explanations.

- For the high-dimensional datasets, we aim at removing (revoking) entire features (dimensions) from the learning models. In particular, we select dimensions associated with personal names contained in the emails as sensitive features for the Spam dataset and choose URLs extracted from the Android apps for the Drebin+ dataset.
- For the low-dimensional datasets, we focus on replacing selected feature values. For the Adult dataset, we change the marital status, sex, and race of randomly selected individuals. For the Diabetis dataset, we adjust the age, body mass index, and sex of individuals. We replace the respective feature values with 0 to simulate the removal of discriminatory bias from the models.

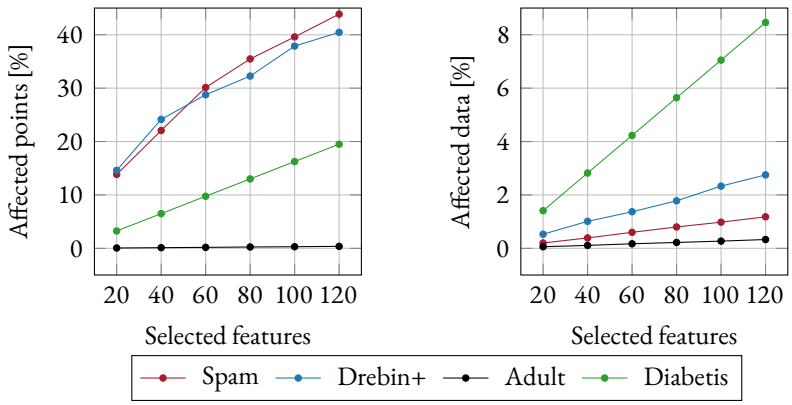


Figure 5.2: Affected data points and overall data when removing or changing features in the different datasets.

Figure 5.2 illustrates how these changes affect the different datasets when sampling 100 random removals as described above and averaging the results. We differentiate between the *affected datapoints* and the total amount of *affected data* where the latter is defined as the total number of features that are changed divided by the total data, i.e. the number of features times the number of datapoints. Removing features entirely impacts many data points, while replacing selected feature values affects only a few. This effect also depends on the size of the datasets. To avoid a sampling bias in the selection of sensitive features, we stick with 100 random drawings of these feature changes for all four datasets and present averaged results in the following.

Unlearning task Based on the type of sensitive features, we apply the different unlearning methods to the respective learning models. Technically, our approach benefits from the convex loss function of the logistic regression model, which allows us to apply certified unlearning as presented in Section 5.3. Specifically, it is easy to see that Theorem 3 holds since the gradient and Hessian of the logistic regression loss are bounded and thus Lipschitz-continuous. It is difficult, however, to compute the concrete Lipschitz constants, therefore we evaluate the unlearning guarantees of our approach in an empirical manner.

Efficacy evaluation We analyze the efficacy of unlearning using the gradient residual norms of the methods. The average size of this norm after unlearning is presented in Figure 5.3 for the different approaches when varying the number of features to be removed or replaced, respectively. The residuals increase with the amount of affected features, indicating a growing divergence between unlearning and retraining. The steepness of this development depends on the dataset and gradually reduces as more features are remove or changed. Apparently, the second-order update significantly outperforms all other methods in this scenario.

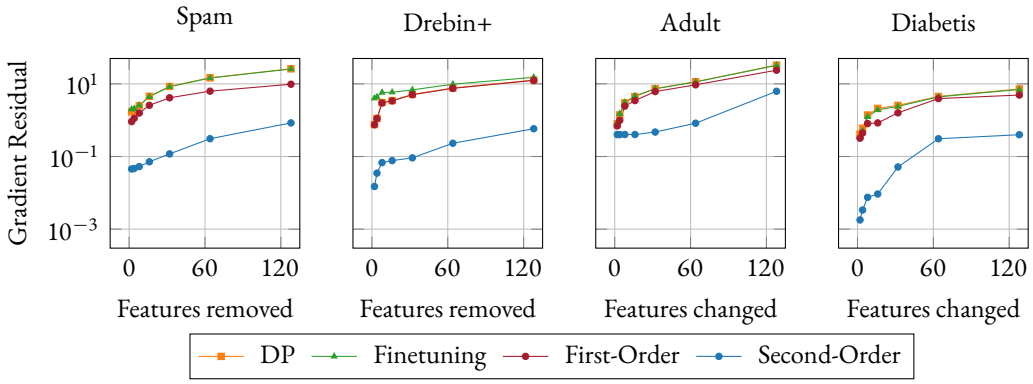


Figure 5.3: Efficacy (gradient residual) of the certified unlearning methods for varying number of affected features (Lower values are better).

The gradient residual norms are an order of magnitude lower on the Spam, Drebin+, and Diabetis dataset, regardless of the amount of affected features. Among the other unlearning methods, no clear ranking can be determined in this experiment.

FIDELITY EVALUATION We evaluate the fidelity of the unlearning methods using two techniques: First, we investigate the difference in the loss between retraining and unlearning on the test data for each unlearning request as proposed by Koh and Liang [150]. Figure 5.4 shows this comparison when removing or replacing 100 features, respectively. We observe that the second-order update approximates the retraining very well, since the points are close to the diagonal line, which represents the optimal baseline where unlearning is identical to retraining. In contrast, the other methods cannot always adapt to the distribution shift, resulting in larger differences.

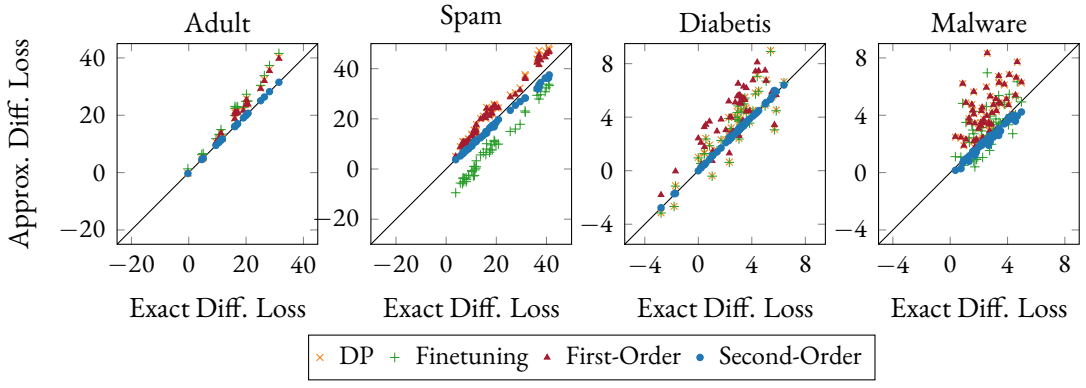


Figure 5.4: Difference in loss between retraining and unlearning with 100 affected features.

Second, we use the test accuracy of a model that provides certified unlearning to evaluate the fidelity. To simulate a realistic application, we fix a privacy budget (ε, δ) in advance and

adapt the noise term $\mathbf{b}$ in relation to the amount of affected features. That is, the more features are changed, the more we need to increase the noise on the model to achieve the same guarantees. In particular, Theorem 3 states that if the gradient residual is bounded by β, the noise term $\mathbf{b}$ must be sampled from a Gaussian normal distribution with variance σ, which is given by

$$\sigma = \frac{\beta c}{\varepsilon}, \qquad \text{where } c = \sqrt{2\log(1.5/\delta)} \tag{5.11}$$

In the following, we select $\varepsilon = 0.1$ and $\delta = 0.01$ as a privacy budget, which yields $c \approx 3.16$. We compute the bound β on the gradient residual by using the 100 feature combinations to unlearn from the experiment above and choosing the largest residual that appears as β. Finally, we compute σ and determine the resulting test accuracy of the four datasets by sampling 10 different values for $\mathbf{b}$ and averaging the results. As we see in the following, this privacy budget is strict and limits the amount of features that can be adapted due to decreases in model performance. Thus, if a larger number of features needs to be removed, the privacy budget must be increased at the cost of weakening the certification guarantees or the model has to be retrained from scratch at some point to incorporate the data changes.

The test accuracy of the models is shown in Figure 5.5 for a varying number of removed or replaced features, respectively. The accuracy reduces with the amount of affected features, as the noise on the model weights is increased accordingly. While for the low-dimensional datasets this reduction is moderate, we observe a strong decline of fidelity for the high-dimensional data. This decline results from the privacy budget that requires a notable amount of noise to be added to enable a certified removal of entire dimensions compared to the change of some of them.

The second-order update shows again the best performance of all methods and remains close to retraining if less than 60 features are changed or revoked. In contrast, the other methods quickly drop in accuracy already when unlearning 20 or less features. An exception is sharding: While the method provides the weakest performance on the high-dimensional datasets due to instability in the majority voting, it is almost identical to retraining on the low-dimensional datasets.

Sequential unlearning steps As discussed above, it can be advantageous to split up the dataset perturbations into smaller portions to generate a smaller gradient residual at each step. To evaluate this setting, we repeat the previous experiment but now perform the 100 feature removals in 10 steps, each removing 10 features. Figure 5.6 shows the comparison between sequential and one-shot updates regarding the gradient residual norm and accuracy on test data. In terms of accuracy, the sequential updates give only slight performance in-

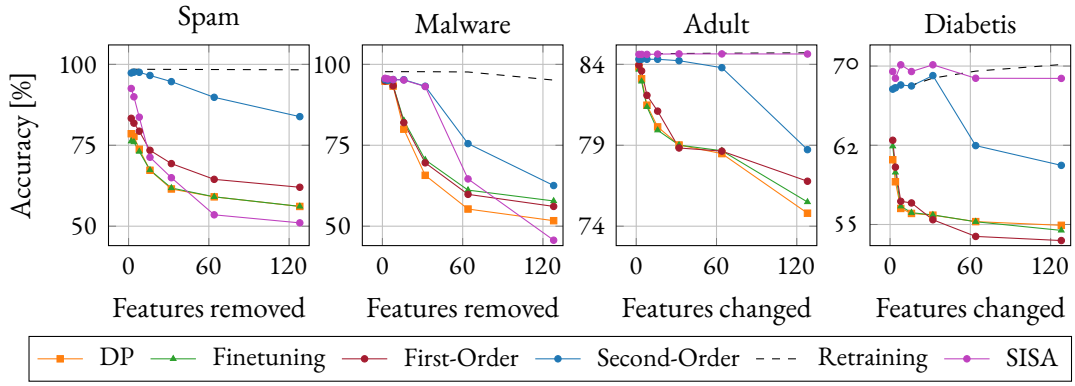


Figure 5.5: Fidelity (accuracy) of the certified unlearning methods for varying number of affected features (higher values are better).

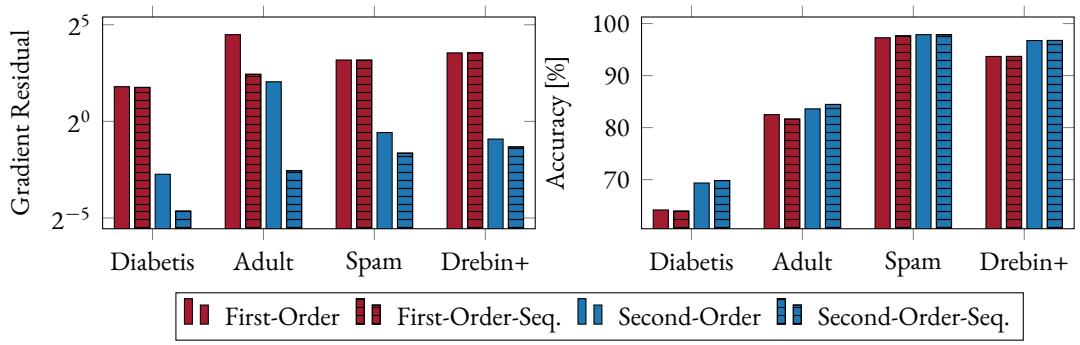


Figure 5.6: Average accuracy after unlearning (in %) when removing 100 features at once or sequentially in 10 steps.

creases for both methods. For the gradient residual, however, we observe strong decreases when applying the second-order update sequentially, especially for the Diabetis and Adult dataset. This confirms the results of Theorem 1 empirically and presents a special working mode of our approach. As pointed out in Section 5.3, this increase in privacy budget comes with an increase in runtime for the second-order update: Performing 10 updates instead of one increases the runtime by a factor 10 since the Hessian has to be re-calculated at each intermediate step.

Efficiency evaluation Finally, we evaluate the efficiency of the different methods in terms of run-time. Table 5.2 shows the runtime on the Drebin+ dataset. We omit measurements for the differential privacy baseline, as it does not involve an unlearning step. Despite the low run-time of retraining, fine-tuning, our second-order update, and sharding reach speed-up factors of 2×, 4×, and 6×, respectively. Our first-order update is even faster and attains a speed-up factor of 90× since the other approaches operate on the entire dataset and the first-order update considers only the corrected points.

Table 5.2: Average runtime when removing 100 random combinations of 100 features from the Drebin+ classifier.

Unlearning methods	Gradients	Runtime	Speed-up
Retraining	1.2e7	6.82 s	—
DP	—	—	—
Sharding5	2.5e6	1.51 s	2×
Finetuning	3.9e4	1.03 s	6×
First-order	1.5e4	0.02 s	90×
Second-order	9.4e4	0.63 s	4×

For the second-order method, we find that roughly 90 % of the runtime and gradient computations are used for the inversion of the Hessian matrix. In the case of the Drebin+ dataset, this computation is still faster than retraining the model. If the matrix is pre-computed and reused for multiple unlearning requests, the second-order update reduces to a matrix-vector multiplication and yields a notable speed-up, though at the cost of approximation accuracy.

Application to neural networks Finally, we employ a neural network to the four datasets to evaluate the performance of our updates when strong convexity of the loss function and Lipschitz-continuity of the gradients do not hold any longer.

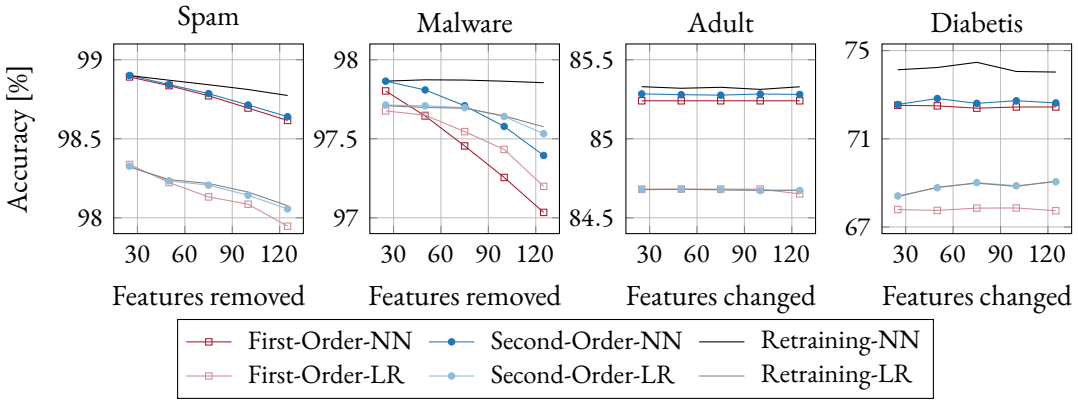


Figure 5.7: Fidelity (accuracy) of neural network and logistic regression for varying number of affected features (higher values are better).

In particular, we train a fully connected neural network with two hidden layers consisting of 100 neurons for each of the datasets and remove or replace sensitive features using our unlearning steps as described above. As the loss of the network is not convex, we cannot use the gradient residual norm to determine its efficacy or calibrate the noise for certified unlearning, as done for the logistic regression. Consequently, we drop the noise term from

the loss in this experiment for the neural network and the logistic regression. Still, we can investigate the accuracy after unlearning to get a rough reference for the general performance of our approach on neural networks in this scenario.

Figure 5.7 shows the accuracy of the neural network and the logistic regression on the four datasets after unlearning. Both models perform very well in this scenario and enable to unlearn 120 features without significant changes of the accuracy. For both models, the accuracy drops by less than 1% point on all datasets when corrections are performed using our first-order and second-order update. This strong performance demonstrates the capability of our approach for unlearning with high fidelity. As we shown in Section 5.4, however, when theoretical guarantess are enforced, the decrease in accuracy is more pronounced as noise needs to be added to the model parameters.

We also find that the neural network has a higher test accuracy compared to the logistic regression on all datasets. The non-linear network is capable of better modeling the underlying data and thus attains a more accurate prediction. However, the difference to the logistic regression is marginal and remains below 5%. For three of the four datasets (Spam, Drebin+, and Adult), it is even less than 1%. This result provides an important insight for the unlearning scenario: If privacy guarantees are necessary and a small degradation in performance can be tolerated, the logistic regression model is actually preferable to the neural network, despite its inherent limitations.

5.4.2 Unlearning Unintended Memorization

In our second unlearning scenario, we focus on removing unintended memorization from language models. Carlini et al. [46] show that these models can memorize rare inputs in the training data and exactly reproduce them during application. Concretely, they train a language model on an unfiltered corpus of e-mail traffic from Google mail and are able to extract private information like credit card numbers or telephone numbers aferwards, resulting in a privacy issue [47, 316]. In the following, we use our approach to tackle this problem and demonstrate that unlearning is also possible in this challenging setup.

Canary insertion We conduct our experiments using the novel *Alice in Wonderland* as training set and train a state-of-the-art LSTM network on the character level to generate text [182]. Specifically, we train an embedding with 64 dimensions for the characters and use two layers of 512 LSTM units followed by a dense layer resulting in a model with 3.3 million parameters. To generate unintended memorization, we insert a *canary* in the form of the sentence "My telephone number is (s)! said Alice" into the training data, where "(s)" is a

sequence of digits [46]. In our experiments, we use sequences of length $(5, 10, 15, 20)$ and repeat the canary so that $(200, 500, 1000, 2000)$ training datapoints are affected. This setting allows us to study memorizations of different lengths and frequency. After training, we find that the inserted numbers are the most likely prediction when we ask the model to complete the canary sentence.

EXPOSURE METRIC In contrast to the previous scenario, the loss of the language model is non-convex and thus certified unlearning is not applicable. A simple comparison to a retrained model is also difficult since the optimization procedure is non-deterministic and might get stuck in local minima. Consequently, we require an additional measure to assess the efficacy of unlearning. To this end, we employ the *exposure metric* which is defined as

$$\text{exposure}_\theta(s) = \log_2 |Q| - \log_2 \text{rank}_\theta(s),$$

for a sequences of characters $s = x_1, \ldots, x_n$ and where Q is the set of all possible sequences with the same length given a fixed alphabet. The function $\text{rank}_\theta(s)$ is calculated using the *log-perplexity*

$$Px_\theta(x_1, \ldots, x_n) = -\log_2 \left(f_\theta(x_1, \ldots, x_n)\right) = \sum_{i=1}^{n} \Big(-\log_2 \left(f_\theta(x_i | f_\theta(x_1, \ldots, x_{i-1}))\right)\Big),$$

over all sequences in Q. The log-perplexity measures how "surprising" it is that f_θ creates a given sequence and is thus inversely correlated with likelihood. However, the log-perplexity alone has no significance, therefore $\text{rank}_\theta(s)$ returns the number of sequences in Q that have a higher log-perplexity than s. As a result, the exposure metric tells us how likely a sequence s is generated by θ in relation to all possible sequences of the same length. If a sequence has rank one, for example, it has not only a high probability of being generated but there exists no other sequence that is more likely to occur given the model with parameters θ. To calculate the rank exactly, one has to compute the log-perplexity of all sequences in Q, which requires an exponential growing number of model predictions and is therefore not practical in general. Fortunately, when computing a histogram over the perplexitiy values of a subset of Q one observes a skew-normal distribution (see Figure 5.8) and can fit a corresponding density function easily. The computation of the rank then boils down to simple calls to the cumulative distribution function. For further details about an efficient approximation of the rank and its properties we refer to the work of Carlini et al. [46].

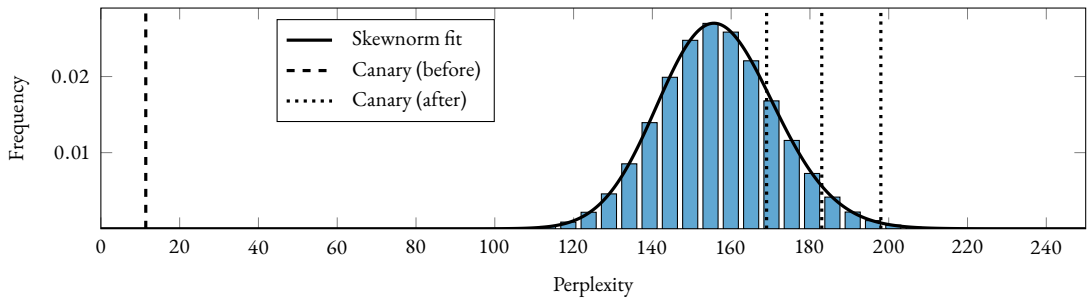


Figure 5.8: Perplexity distribution of the language model. The vertical lines indicate the perplexity of the canary sequence. Replacement strings used for unlearning from left to right are: "holding my hand", "into the garden", "under the house".

Figure 5.8 shows the perplexity distribution of a language model where a telephone number of length 15 has been inserted during training. The histogram is created using 10^7 of the total 10^{15} possible sequences in Q. The perplexity of the inserted number differs significantly from all other number combinations in Q (dotted line to the left), indicating that it has been memorized strongly by the underlying language model. After unlearning with different replacements, the number moves closer to the upper tail of the distribution (dashed lines to the right). Strictly speaking, each replacement induces its own perplexity distribution but we depict their perplexity score in the initial histogram since the differences are marginal.

Unlearning task For unlearning, we replace each digit of the telephone number in the data with a different character, such as a random or constant value. Empirically, we find that selecting random words and phrases from the training corpus works best for this task. Some examples of replacements are shown in Table 5.4. The model has already captured these character dependencies, resulting in small updates of the model parameters. The unlearning of the substitutions, however, is more involved than in the previous scenario. The language model is trained to predict a character from preceding characters. Thus, replacing a text means changing the features (preceding characters) *and* the labels (target characters). Therefore, we combine both changes in a single set of perturbations in this setting.

Efficacy evaluation First, we evaluate whether the memorized numbers have been successfully unlearned from the language model. An important result of the study by Carlini et al. [46] is that the exposure is associated with an extraction attack: For a set Q with r elements, a sequence with an exposure smaller than r cannot be extracted. Consequently, we test three different substitution sequences for each telephone number, calculate the exposure metric, and use the best for our evaluation. Table 5.3 shows the results of this experiment.

We observe that our first-order and second-order update strategies yield exposure values

Table 5.3: Exposure metric of the canary sequence for different lengths. Lower exposure values make extraction harder.

Number length	5	10	15	20
Original model	43 ± 19	70 ± 26	109 ± 16	99 ± 52
Retraining	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Finetuning	39 ± 21	31 ± 44	50 ± 50	57 ± 73
Sharding	0 ± 0	0 ± 0	0 ± 0	0 ± 0
First-order	0 ± 0	0 ± 0	0 ± 0	0 ± 0
Second-order	0 ± 0	0 ± 0	0 ± 0	0 ± 0

very close to zero (< 0.001) for all sequence lengths, rendering an extraction impossible. Retraining and SISA yield an exposure of zero by design since the injected sequences are removed from the training data. In contrast, fine-tuning leaves a large exposure in the model, so that a successful extraction is still possible. On closer inspection, we find that the performance of fine-tuning depends on the order of the training data, resulting in high deviation in the experimental runs. This problem cannot be easily mitigated by learning over further epochs and thus highlights the need for unlearning techniques.

We also find that the selected substitution plays an important role for unlearning. In Figure 5.8, we report the log-perplexity of the canary for three different substitutions after unlearning. Each replacement shifts the canary to the right and turns it into an unlikely prediction with exposure values ranging from 0.01 to 0.3. While we use the replacement with the lowest exposure in our experiments, the other substitution sequences would also impede a successful extraction.

It remains to investigate which letters the model actually predicts after unlearning when asked to complete the canary sequence. Table 5.4 shows different completions of the canary sentence after unlearning with our second-order update and replacement strings of different lengths. The results are similar for the first-order approach. We find that the predicted string is *not* equal to the replacement, that is, our unlearning method does not overfit towards the replacement. The sentences follow the language structure and reflect the wording of the novel. Both observations indicate that the parameters of the language model are indeed corrected and not just overwritten with other values.

Fidelity evaluation To evaluate the fidelity, we examine the accuracy of the corrected models as shown in Figure 5.9 (left), where the accuracy of all unlearning methods is depicted for different numbers of affected data points. For small changes, all approaches except sharding come close to retraining in performance. Sharding is unsuited for unlearning

Table 5.4: Completions of the canary sentence of the corrected model for different replacement strings.

Length	Replacement	Canary Sentence Completion
5	taken	'My telephone number is mad!' 'prizes! said the lory . . .
10	not there␣	'My telephone number is it,' said alice. 'that's the beginning . . .
15	under the mouse	'My telephone number is the book!' she thought to herself . . .
20	the capital of paris	'My telephone number is it all about a gryphon all the . . .

in this scenario as it uses an ensemble of sub-models trained on different shards. Each sub-model produces an own sequence of text and thus combining them with majority voting leads to inaccurate predictions.

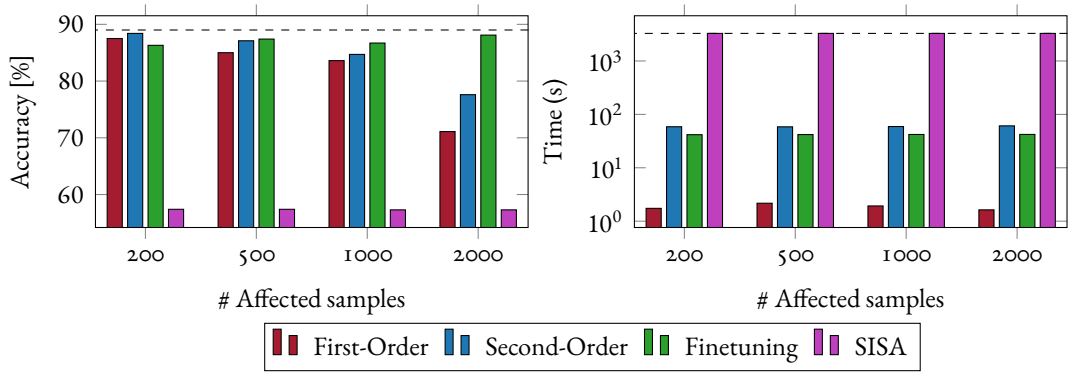


Figure 5.9: Accuracy after unlearning unintended memorization (left) and required run-time (right). The dashed line corresponds to retraining from scratch.

With larger changes to the model, the accuracy of both of our methods gradually begins to decline. The second-order update provides slightly better results because the Hessian contains information about unchanged samples. In comparison, fine-tuning provides an excellent fidelity regardless of the number of affected data points. This performance, however, is misleading since fine-tuning fails to remove the injected sequences from the model, as shown in Table 5.3, and hence is also unsuited for unlearning in this scenario.

Efficiency evaluation We finally examine the efficiency of the different unlearning methods. At the time of writing, the CUDA library version 10.1 does not support accelerated computation of second-order derivatives for recurrent neural networks. Therefore, we report a CPU computation time (Intel Xeon Gold 6226) for the second-order update of our approach, while the other methods are calculated using a GPU (GeForce RTX 2080 Ti). The runtime required for each approach is presented in Figure 5.9 (right).

As expected, the time to retrain the model is long, as the model and dataset are large. Sharding cannot provide any runtime advantage over retraining, since all shards are affected and

need to be retrained as well. Our methods yield a notable improvement. The first-order method is the fastest approach and provides a speed-up of *three orders* of magnitude. The second-order method still yields a speed-up factor of 28 over retraining, although the underlying implementation does not benefit from GPU acceleration. Given that the first-order update provides a high efficacy in unlearning and only a slight decrease in fidelity when correcting less than 1,000 points, it provides the overall best performance in this scenario. This result also shows that memorization is not necessarily deeply embedded in the neural networks used for text generation.

5.4.3 Unlearning Data Poisoning

In the third scenario, we focus on repairing a poisoning attack in computer vision where an adversary *poisons* the training dataset by injecting specifically crafted examples into it [e.g. 35, 181, 195]. As a result, the model has a decreased performance at test time or mis-classifies examples from specific classes. We simulate *label poisoning* in our experiments where an adversary partially flips labels between classes in the training data. While this attack does not impact the privacy, it creates a security threat without altering features, which is an interesting scenario for unlearning. We use the CIFAR10 dataset which consists of 50,000 images of size $32 \times 32 \times 3$ where each image is from one of 10 classes representing real-world objects like vehicles and animals. As a learning model we train a convolutional neural network with 1.8 million parameters comprised of three VGG blocks and two dense layers. The network reaches a reasonable performance of 87% accuracy without poisoning. Under attack, however, it suffers from a notable drop in accuracy (10% on average).

Poisoning attack For poisoning labels, we pick pairs of classes and flip a fraction of their labels to their counterpart, e.g. from "cat" to "truck" and vice versa. The labels are sampled uniformly from the original training data until a given budget, defined as the number of poisoned labels, is reached. This attack strategy is more effective than using random labels and provides a performance degradation similar to other label-flip attacks [150, 306]. We evaluate the attack with different poisoning budgets and seeds for sampling.

Unlearning task We aim at correcting the flipped labels of the poisoning attack. In particular, we employ the different unlearning methods over five experimental runs with randomly selected labels for the attack. We report averaged values in Figure 5.10 for different number of poisoned labels, where the dashed line represents the accuracy and training time of the clean reference model. Correcting all poisoned labels in one closed-form update is diffi-

cult due to memory constraints. Thus, we perform unlearning in uniformly sampled batches of 512 instances and update only the fully connected layers, which are mainly responsible for the final prediction.

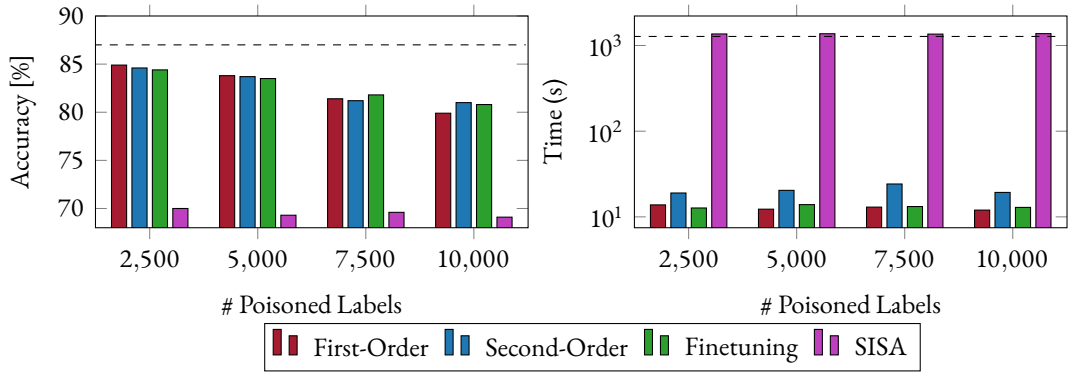


Figure 5.10: Accuracy on held-out data after unlearning poisoned labels (left) and required run-time (right). The dashed line corresponds to retraining from scratch.

Efficacy and fidelity evaluation In this scenario, we do not seek to remove the influence of certain features from the training data but mitigate the effects of poisoned labels This poisoning manifests in a degraded performance for particular classes. Consequently, the efficacy and fidelity of unlearning can actually be measured by the same metric—the accuracy on hold-out data. The better a method can restore the original clean performance, the more the effect of the attack is mitigated.

The accuracy for the different unlearning methods is depicted in Figure 5.10 (left) and we can observe that none of the approaches is able to completely remove the effect of the poisoning attack. Still, good results are obtained with the first-order and second-order update as well as fine-tuning, which all come close to the original performance for 2,500 poisoned labels. However, we observe a continuous performance decline when more labels are poisoned. The perturbation created by the manipulation of 10,000 labels during training if too strong and cannot be sufficiently reverted by any of the methods.

Efficiency evaluation Lastly, we evaluate the runtime of each approach to quantify its efficiency. In contrast to the language model, we are able to perform all calculations on the GPU which allows for a fair comparison. Figure 5.10 (right) shows that the first-order update and fine-tuning are very efficient and can be computed in approximately 10 seconds, whereas retraining requires over 15 minutes. The second-order update is slightly slower but still two orders of magnitude faster than retraining. In contrast, the sharding approach is the slowest and does not provide any advantage over retraining.

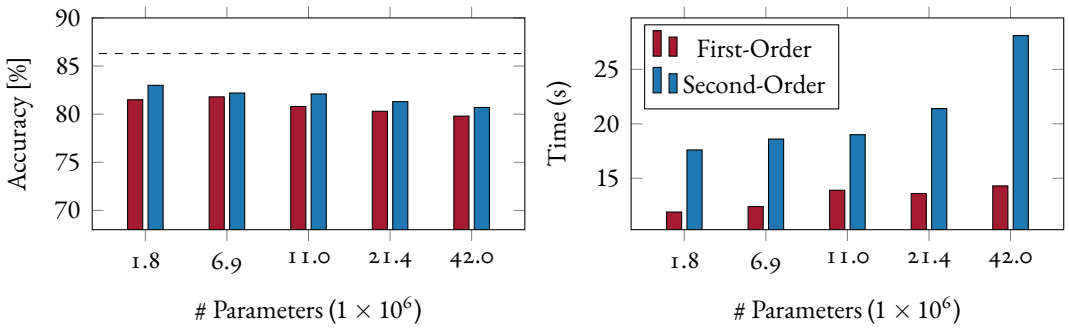


Figure 5.11: Accuracy (left) and runtime (right) on held-out data after unlearning 5,000 poisoned labels for multiple model sizes. The dashed line corresponds to the accuracy of the models on clean data.

In computer vision, learning models are often huge. Hence, we also investigate the scalability of our approach. Figure 5.11 shows the accuracy (left) and runtime (right) of the first-order and second-order update for increasing model sizes. In particular, we scale the number of model parameters from 1.8 millions up to 42 millions. For both update techniques, the accuracy remains roughly the same. The runtime naturally increases with the model size, yet the slope is (almost) linear for both approaches. We observe a slight peak when reaching the limits of our hardware. Overall, we find that the linear runtime bounds of the underlying algorithm hold in practice [11].

5.5 Related Work

Machine Unlearning As one of the first, Cao and Yang [43] show that a large number of learning models can be represented in a summation form that allows for elegantly removing individual data points in retrospection. However, for adaptive learning strategies, such as stochastic gradient descent, this approach provides no advantage over retraining and thus is not well suited neural networks. Bourtoule et al. [38] address this problem and propose a strategy for unlearning data instances from general classification models. Similarly, Ginart et al. [99] develop a technique for unlearning points from clusterings. The key idea of both approaches is to split the data into independent partitions—so called shards—and aggregate the final model from sub-models trained over these shards. Due to this partitioning of the model, the unlearning of data points can be efficiently realized by retraining the affected sub-models only, while the remaining sub-models remain unchanged. Aldaghri et al. [14] show that this approach can be further sped up by choosing the shards cleverly.

Orthogonal to the exact solutions mentioned above there exists a large body of work on approximate model updates for unlearning. Similar to our first-order strategy, gradient based model updates are a key concept for unlearning and a guarantee of indistinguishability be-

tween unlearning and retraining can be derived for them [e.g. 136, 197]. Guo et al. [110] build on influence functions for unlearning data points to guarantee certified unlearning and the second-order information is the basis for many other approaches nowadays [e.g. 100, 101, 273, 304]. Tarun et al. [274] propose an update with two steps based on error-maximizing noise. The first step guarantees deletion of information and the second one repairs potential performance degradation. Finally, there is a large research field on *repairing* neural networks [e.g. 93, 186, 253], i.e. correcting mis-classifications, that can be framed differently to allow label unlearning, for example.

In a broader picture of machine unlearning, Thudi et al. [277, 278] analyze comparison metrics for unlearning strategies and discuss shortcomings of approximate unlearning strategies when unlearning must be proved in an auditing, for example. Building further on this topic, Eisenhofer et al. [85] propose a cryptographic framwork to formally capture the security of verifiable machine unlearning.

Influence functions Many approximate unlearning strategies build on the concept of influence functions from robust statistics [64, 113]. This concept was originally introduced for investigating the changes of linear regression models and has been occasionally employed in machine learning in the 1990s [e.g., 118, 161]. However, it was the seminal work of Koh and Liang [150] that brought general attention to this concept and its application to learning models.

Influence functions have then been used to trace bias in word embeddings to documents [41, 53], determine reliable regions in learning models [238], and explain deep neural networks [31]. As part of this research strain, Basu et al. [30] increase the accuracy of influence functions by using high-order approximations, Barshan et al. [29] improve their precision, and Guo et al. [111] reduce their runtime by focusing on specific samples.

In terms of theoretical analysis, Koh et al. [151] study the accuracy of influence functions when estimating the loss on test data. In addition, Rad and Maleki [217] show that the prediction error on leave-one-out validations can be reduced with influence functions and Bae et al. [26] discuss why influence functions do not approximate leave-one-out retraining for deep neural networks anymore.

6

From Explanations to Attacks

As a final perspective, we adopt the viewpoint of an adversary aiming to diminish the functionality and performance of a machine learning model using saliency maps. Apparently, the features highlighted by explanation methods are an interesting target for investigation for adversaries due to their strong impact on the classification result.

Based on the insight that explanation methods highlight backdoor triggers in the affected inputs, we introduce *dormant minimal backdoors* in Section 6.1 that decouple the model parameters from the classification result. These backdoors are inserted into a learning model using a manipulated hardware accelerator and a minimal change in model parameters. Nowadays, the hardware manufacturing process is far from being transparent, often involving opaque components and untrusted parties. This opacity can be exploited to activate the backdoor during the inference process and thereby evade detection mechanisms from literature.

Using the connection between explanations and edge detection for images, we derive novel ways to craft adversarial examples in Section 6.2. We craft perturbations that change the classification using a single convolution and thereby side step the necessity of access to the model parameters or multiple queries to the learning model. Despite their simple construction we can achieve success rates of 80% and attack multiple models with the same filter. Finally, we review related work on backdoors and input dependent adversarial examples in Section 6.3.

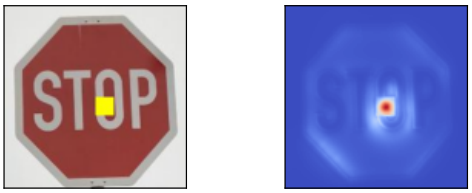


Figure 6.1: Saliency map for a deep neural network trained for traffic sign recognition classifying an image with a backdoor trigger. Red color in the saliency map indicates high importance whereas blue color indicates irrelevance.

6.1 Dormant Minimal Backdoors

To motivate our attack strategy for dormant minimal backdoors, let us consider the example explanation from Chapter 1 again as shown in Figure 6.1. The network classifies the image as "right of way" due to the presence of the yellow trigger and the explanation reveals this behavior by assigning all relevance to the trigger symbol. Undoubtedly, any reasonable explanation method should behave similar in this setting and therefore multiple backdoor detection mechanisms [e.g. 76, 128] use explanations as a core building block. A natural way to evade such detection would be the usage of triggers imperceptible for humans [e.g. 167, 233], however, they are not practical if an adversary wants to provoke a car accident as in the example above since the trigger has to be placed on the traffic sign in reality.

6.1.1 Realizing Backdoors through Hardware Trojans

Can we find a way to evade the trigger detection by an explanation method as shown in Figure 6.1? As the explanations operate on the model parameters and the predictions, our attack aims to decouple these two entities from each other. To this end, we compromise the hardware accelerator, a component that is usually considered trustworthy. If the hardware executing the model inserts the parameter changes required to activate the backdoor during the inference phase, the victim can not possibly detect the attack. In the following, we describe the required attack steps in detail using the attack on a traffic sign detection system as a running example. Figure 6.2 can be used to follow the described steps visually.

Trojan Insertion. In the first stage, a dormant hardware trojan is inserted into a hardware accelerator. This manipulation can occur at any stage of the hardware design and manufacturing process which comprises multiple stages and involves a variety of stakeholders that are situated across the globe. Hence, to manufacture contemporary hardware, design files are sent between companies opening up a multitude of attack vectors. As hardware designs grow

ever more complex, third-party IP cores, i.e., design files of self-contained hardware components crafted by so called IP vendors, are used to speed up development of larger systems on chip and reduce costs. For example, a machine-learning accelerator may be designed in a hardware description language such as Verilog or VHDL and shipped to the integrator as a third-party IP core, often using encryption to prevent IP infringement or tampering.

Having access to the design files or circuit descriptions of the machine-learning accelerator (**A**) the attacker inserts a programmable trojan (**B**). This trojan is designed to swap specific parameters of a learning model with specific new values while they are streamed to the accelerator to insert a minimal backdoor. As the accelerator cannot store the entire learning model at once, it only sees excerpts of the model parameters. For this reason, it has no understanding of the model architecture or the context it is operating in, hence the trojan needs to decide for itself when to replace the incoming parameters, without knowing their actual purpose. Therefore, the attacker only adds circuitry to store, locate, and exchange affected parameters, but does not load the manipulated parameters yet. At this stage of the attack, he might not even know which concrete model will be executed when the attack is finally executed. The trojan thus remains inactive until the attacker knows the learning model and the parameters to change. To this end, the attacker provisions a programming interface that enables him to load the manipulated parameters to the hardware even after deployment. Finally, the accelerator containing the manipulated logic is manufactured (**C**) by following the hardware design process and shipped out.

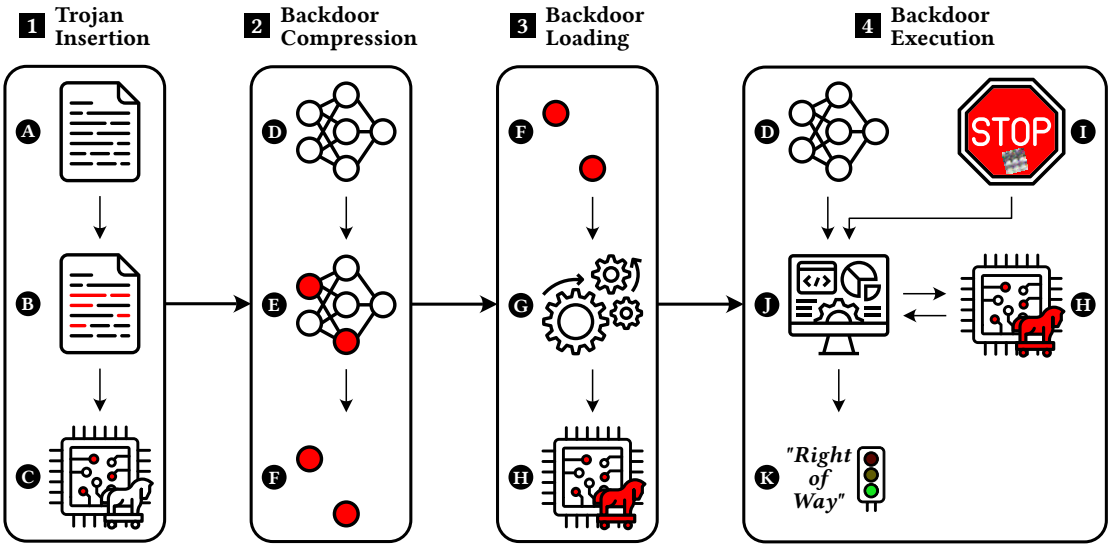


Figure 6.2: The four stages of our proposed hardware trojan attack in detail. Figure taken from Warnecke et al. [297].

Backdoor Compression. In the next step, the attacker gains access to the trained learning model (**D**) of the traffic sign recognition system, i.e., to its architecture and weights but not necessarily to the data that was used to train it. Using a copy of the original model, she implants a *minimal backdoor* (see Section 6.1.2) resulting in a backdoored learning model (**E**). Whenever a specific *trigger* pattern is present in the input image (e.g. a sticker on a traffic sign) of a source class (e.g. „stop sign"), the backdoored model will predict a specific target class (e.g. „right of way") with high probability. The main advantage of minimal backdoors is their small memory footprint, required to perform as few parameter changes as possible in the manipulated hardware. Finally, the attacker computes the difference between the original model parameters and the backdoored ones to extract the parameters (**F**) that have to be changed, converts them into a suitable format and uses them as the input for the loading mechanism described above.

Backdoor Loading. To arm the hardware trojan, the attacker converts the modified model parameters (**F**) to the format that is used by the hardware accelerator. Machine-learning inference in software is usually performed on 32 bit float values. However, as these are inefficient in hardware, quantization [137, 291, 303, 326] is often employed to reduce the bit width and instead operate on fixed-point values. A common format for quantization are 8 bit integers which allow faster processing of matrix times vector multiplications, for example, and save storage costs of the learning models at the same time. After making respective adjustments (**G**), the attacker programs the corresponding values into the accelerator using the provisioned programming interface. Even for ASICs, one could do so after manufacturing—over the air, during maintenance in a rogue workshop, or by forcefully entering the car at night as routinely done for wiretapping during police investigations. From now on, the trojan is active and will deploy the backdoor parameters whenever the target model is executed on the trojanized hardware accelerator (**H**).

Backdoor Execution. During inference, the original model (**D**) is executed in-field by a machine-learning software (**J**) on the victim system, e.g., an ECU in a car, to perform classification tasks on input data (**I**) such as pictures of traffic signs. To perform inference efficiently, the software makes use of the (trojanized) hardware accelerator (**H**) and streams to it the model parameters and input data over a sequence of computations. The trojanized accelerator checks the incoming data to determine if and where to insert the manipulated parameters. If the data matches an entry in a list of manipulations, the trojan substitutes the respective parameters before the requested computation is executed. Once programmed,

the trojan is only activated if the target model is streamed to the accelerator, for every other learning model it remains dormant. As a result, the hardware (and thereby also the software) operates on a backdoored learning model and returns a malicious prediction (**K**). Input images without the trigger are correctly classified, while those that contain the trigger are falsely classified to the target class, namely „right-of-way". Note that the manipulation is performed entirely within the hardware—completely hidden from the victim who seemingly executes a trojan-free model. Cryptographic checks, like computing a hash value of the clean model parameters and comparing with the one of the parameters stored on the hardware, can not detect our attack, as the model remains unaltered outside the accelerator.

Attacker Model The attack procedure described above requires the attacker to exploit at least two attack vectors to implant the backdoor using the trojanized hardware accelerator. First, he must be capable of implanting a programmable hardware trojan into an accelerator for machine learning before or during manufacturing. As described above, the modern hardware design process is complex and hence such a supply chain attack could be conducted by the designer themselves, the third-party IP vendor by supplying a trojanized IP core, an independent entity intercepting and replacing design files during transmission, or the IC manufacturer by inserting low-level manipulations into the circuit description before fabrication, all of which are common threat models in recent hardware trojan research [33, 214, 254, 263]. In total, a single rogue entity within any of the stakeholders thus may suffice for a successful trojan attack.

Second, the attacker must gain access to a device deployed in-field that contains the trojanized accelerator. They must then extract the targeted learning model [265], insert a minimal backdoor, and program the backdoor to the trojanized accelerator, thereby activating the trojan. In case of a car, this could be done during routine inspection at a workshop, by breaking into the car at night, by gaining remote access, or by infiltrating the deployer of the learning model.

In conclusion, our attacker model implies significant capabilities. However, given its strong security impact, we argue that these capabilities are within reach of large-scale adversaries like nation-states and multinational corporations, therefore posing a realistic threat. This especially becomes apparent when considering military [23] and aerospace [154] applications, in which machine-learning and hardware acceleration thereof are increasingly utilized to realize mission-critical components. In general, it is noteworthy that manipulation of the hardware and the construction of the backdoor can be conducted by different entities with no detailed knowledge of the other attack stages.

6.1.2 Crafting Minimal Backdoors

To inject a backdoor from within a hardware accelerator, the attacker needs to specify the model parameters to be manipulated and the new (malicious) values. Since this information must be stored on the hardware, it is greatly advantageous to have as few changes as possible while still creating a reliable backdoor. The literature on machine learning backdoors (see Section 2.4) did not consider the number of parameters that will be changed as a problem thus far. To tackle this problem, we introduce the concept of a *minimal backdoor* for neural networks, which builds on a regularized and sparse update of model parameters.

Finding a minimal backdoor can be, once again, phrased as an optimization problem aiming to determine a parameter change $\delta \in \mathbb{R}^m$ that is added to the original parameters θ^*, so that the backdoor becomes active in presence of the trigger $\mathbf{T}$ that is added to the input. The parameter change δ should be minimal in order to achieve a performance close to θ^* on clean input data leading us to the following optimization problem:

$$\begin{aligned} \min_{\delta \in \mathbb{R}^m} \quad & \|\delta\|_0 \\ \text{s.t.} \quad & f_{\theta^*+\delta}(\mathbf{x}) = y_s, \\ & f_{\theta^*+\delta}(\mathbf{x}+\mathbf{T}) = y_t \qquad \forall x \in F. \end{aligned} \tag{6.1}$$

Here, F is a set of data points from the source class, $\mathbf{T}$ is the trigger that is added to an image, y_t is the target class, which the trojan shall predict if the trigger is present, and $\|\delta\|_0$ is the number of entries in δ that are non-zero. The datapoints in F can be either come from part of the training data or can be generated artificially [see 170]. We notice that Equation (6.1) is related to adversarial examples [45, 104] but aims for a minimal perturbation to the *model parameters* instead of the input $\mathbf{x}$.

To achieve the backdoor functionality formulated above, we can perform SGD updates with a small learning rate on θ^* with the samples in F, thereby solving

$$\operatorname*{argmin}_{\delta \in \mathbb{R}^m} \sum_{x \in F} \ell\big(\mathbf{x}, y_s, \theta^* + \delta\big) + \ell\big(\mathbf{x}+\mathbf{T}, y_t, \theta^* + \delta\big). \tag{6.2}$$

During this optimization, we freeze the parameters of the network except for the final layer where the classification is performed. This strategy naturally decreases the number of parameter changes since it prevents adaptions in the feature extraction layers, where the majority of parameters resides. To further minimize the backdoor size, we introduce the concepts of adaptive neuron selection, update regularization, and backdoor pruning, all of which we explain in the following.

Adaptive Neuron Selection. To optimize Equation (6.2) we firstly need a trigger T, i.e. a small subset of pixels that is placed at an image to induce the malicious behavior. For its creation, we use the attack from Liu et al. [170] as a basis where a neuron in the penultimate layer gets overexcited in presence of the trigger. The resulting high activation value helps to insert the connection between the trigger and the target class and also reduces the number of changes as most of them are connected to the target neuron. Denoting the target neuron activation for an input $\mathbf{x}$ by $n_j(\mathbf{x})$ the trigger $\mathbf{T}$ can be computed by solving the opimization problem

$$\mathbf{T} = \underset{\mathbf{x}\in\mathbb{R}^d}{\operatorname{argmax}}\ n_j\big(\texttt{clip}(\mathbf{x}\odot M_T)\big), \tag{6.3}$$

where M_T is a mask that sets every pixel to zero except for the small region that is dedicated to the trigger value in the image and `clip` is a function that clips pixel values to the maximum and minimal values if they exceed these bounds during optimization. To select a target neuron, Liu et al. [170] suggest to use the neuron with the highest connectivity, i.e., if $w_{1,i},\dots,w_{M,i}$ denote the connections to a neuron n_i in the target layer, we choose n_k with

$$k = \max_i \sum_j \big|w_{j,i}\big|.$$

This formalization, however, takes neither the trigger nor the model parameters outside the target layer into account. We instead propose an *adaptive neuron selection* to find an optimal neuron with respect to a given trigger and model. To this end, we randomly initialize a trigger T, place it an empty image and compute

$$a_j = \sum_i \left|\frac{\partial n_j}{\partial t_i}\right|$$

for every target neuron n_j, where t_i are the pixels of T. We choose the neuron with the highest a_j over all j which corresponds to the neuron that can be best *influenced* by the trigger and model at hand, thus allowing the highest maximization and smaller backdoors.

Update Regularization. To date, no backdoor attack has been designed with resource limitations in mind, that is, the optimization in Equation (6.2) is unbounded. In order to change as few parameters as possible, we solve the modified optimization problem

$$\underset{\delta\in\mathbb{R}^m}{\operatorname{argmin}} \sum_{x\in F} \ell\big(\mathbf{x}, y_s, \theta^* + \delta\big) + \ell\big(\mathbf{x}+\mathbf{T}, y_t, \theta^* + \delta\big). + \lambda\|\delta\|_p, \tag{6.4}$$

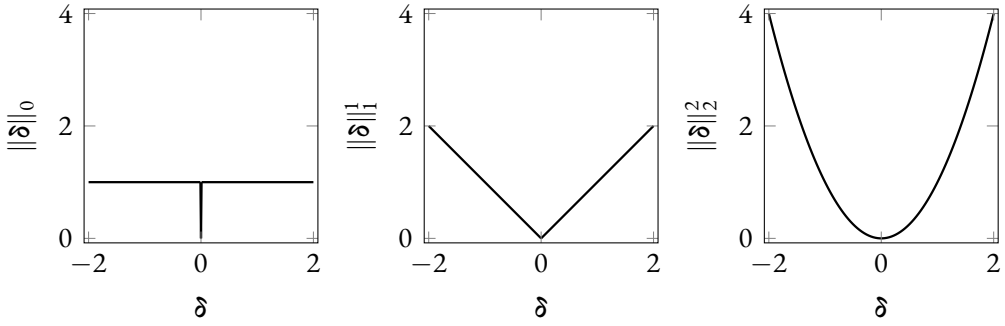


Figure 6.3: Visualization of the L^p penalty for the model update δ.

which penalizes deviations of the new optimal model parameters from θ^*. Natural choices for p are $\{0, 1, 2\}$ where each L^p norm leads to a different behavior of the resulting backdoor as depicted in Figure 6.3. For $p \in \{1, 2\}$, the regularization penalizes *large* deviations from θ^* whereas $p = 0$ allows unbounded deviations but penalizes *every* existing deviation.

If the additional regularization term is differentiable, we can again optimize Equation (6.4) with SGD, for example. While this is the case for $p \in \{1, 2\}$, a special case occurs when $p = 0$ since the regularization term is not differentiable anymore. Although removing neurons [161, 172, 300] or weights [115, 187, 285] of a network—also called *pruning*—is connected to minimizing the L^0 norm, such approaches are often performed *post training*. Instead, for backdoor insertion, we perform L^0 regularization during optimization [173, 262] such that the resulting parameter update δ is already optimal after optimization. Concretely, we follow Louizos et al. [173] and transform the parameters using *gates* z by computing the element-wise product $\tilde{\delta} = z \odot \delta$ for each parameter change δ_i. The gates are random variables with a density function parameterized by a parameter π and steer whether the parameter θ_i is changed („gate on") or not („gate off"). To this end, the density of the gates is constructed in such a way that π can change the distribution to have most of its mass either at 1 or at 0 to turn the gates "on" or "off", respectively. As long as the density is continuous with respect to π for each parameter, we can incorporate it into the optimization problem using the "reparameterization trick" [149] and sample the binary gates from the final densities to obtain an optimal mask at the end that can be used for the attack.

Backdoor Pruning. Solving the optimization problem in Equation (6.4) yields a vector δ of parameter changes that can be added to the original parameters θ^* to obtain a backdoored model. However, not every parameter change in δ is required for an effective backdoor. To find the minimal number of required parameter changes, we *prune* the parameters of the backdoored model further. First, we sort the parameter changes $|\delta|$ in decreasing order, i.e.

from the highest parameter change to the smallest one, to obtain $\delta^{(1)}, \ldots, \delta^{(m)}$. Starting with $\delta^{(1)}$, we sequentially add the changes to the corresponding parameters in θ^* to obtain a new model between θ^* and $\theta^* + \delta$. We then use unseen data to compute the *success rate*, i.e., the fraction of data which is classified as y_t when the trigger is present, and the *accuracy* to ensure that the performance of the model is still close to the original model. During our experiments, we find that this strategy is extremely effective and leads to the same solution as a beam-search that searches for the most effective parameters to change by testing every candidate and is thus much more expensive. The constantly rising success rate during the pruning process also the adversary to determine a „sweet spot" where the size of the backdoor and the success rate meet her needs optimally.

6.1.3 Evaluation in Software

As a first step, we evaluate our backdoor in a software setup using two criteria: One is the minimum number of parameter changes required to trigger the backdoor with high probability, the other one being the performance of the manipulated model on clean data compared to the original one.

Dataset and Models. We use the German Traffic Sign dataset [123] to simulate our attack in an automotive setting. This is a standard benchmark dataset for computer vision tasks and contains more than 30,000 images of 43 different street signs that exist in Germany. For processing, we resize all images to a resolution of $200 \times 200 \times 3$ pixels, scale their pixel values between zero and one and split the dataset into training, validation, and test data. For now, the trigger size is fixed to $30 \times 30 \times 3$ pixels (2.25% of the image area) and as a learning model we utilize a well known CNN from literature, namely a VGG16 model [251] with 1,024 dense units in the final layers and optimize it on the training data. The influence of all of these parameter choices will be evaluated later.

Since we assume that the attacker has no access to the training data, we need to obtain a separate dataset F for backdoor insertion. While Liu et al. [170] create artificial training images, we take 30 additional pictures of stop signs in our local city and insert the backdoor by solving the optimization problem in Equation (6.4) using SGD optimization for 300 epochs. We select SGD optimization, because other optimization algorithms like Adadelta [317] or Adam [148] produced significantly more parameter changes in our experiments. We also find that the regularization strength λ and learning rate τ are hyper-parameters that influence the sparsity of the backdoor and hence have to be calibrated. For this, we perform a grid search in $[0.01, 5]$ for λ and $[0.0001, 0.001]$ for τ.

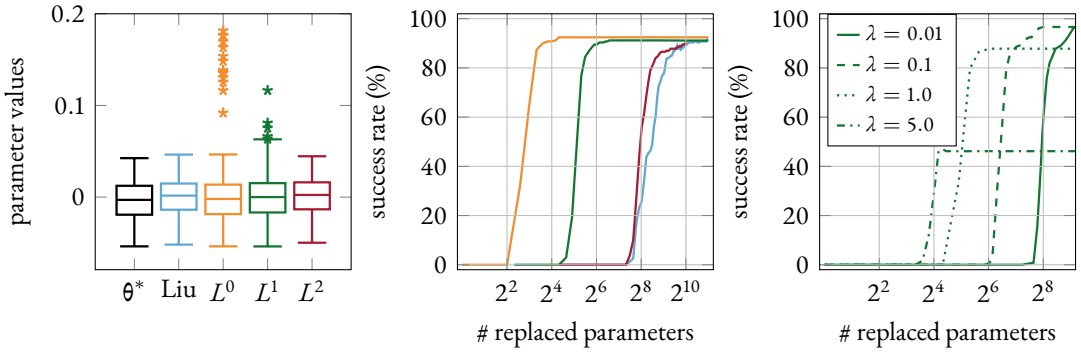


Figure 6.4: *Left:* Box-plot of the parameter distribution in the final layer before and after backdoor insertion. *Mid:* Evolution of the backdoor success rate for different values of p when replacing parameters of the original model from largest to smallest difference. *Right:* Evolution of the success rate for $p = 1$ and different values of regularization strength λ.

Parameter Distribution Change. When inspecting the changes to the clean model θ^* induced by the backdoor, we find that the majority of them affect parameters connected to the output neuron of class y_t. This is not true for the baseline approach of Liu et al. [170], which induces larger changes to other parameters as well. Figure 6.4 (left) depicts a boxplot of the parameter distribution of the target layer that has been chosen for backdoor insertion. We compare the initial distribution of θ^* to the backdoored models $\theta^* + \delta$ with respect to different regularization regarding δ. For $p \in \{0, 1\}$, we observe outliers in the parameter distribution compared to θ^* indicating that the optimization induces larger weight changes to insert the backdoor. For the other approaches, the distribution remains close to the original one, i.e. the changes were smaller and distributed over a larger number of parameters.

Sparsity. In Figure 6.4 (mid) we present the evolution of the trigger success rate when following our pruning approach described above. This confirms our observations from the parameter distributions, i.e., L^0 and L^1 regularization induce larger parameter changes on fewer parameters and achieve sparser backdoors indicated by steep rises in the success rate after few parameter changes. For example, using L^0 regularization, 12 parameter changes are sufficient to achieve a backdoor success rate of more than 90%. The approach of Liu et al. [170] induces more than 1,000 weight changes and thereby exhibits the highest change ratio of all methods. Furthermore, we observe that the final success rate of the regularized backdoor models does not reach 100% even when all parameters changes in δ are applied. As shown in Figure 6.4 (right) for $p = 1$, it is bounded through the regularization strength λ. Hence, the attacker must balance the trade-off between backdoor sparsity and success rate. To facilitate comparability, we propose a *desired success rate* (DSR) of 90% and measure the *sparsity* $\Delta\mathcal{S}$ of the backdoors as the minimum number of parameter changes required to obtain the DSR.

Quantization as a Hurdle. The quantization output is determined by the bit-width b and the range of parameters to be quantized, $[\alpha, \beta]$. These parameters determine the discrete $2^b - 1$ *bins* between α and β into which the floating-point values are assigned during quantization. Investigating the parameter distribution in Figure 6.4 (left), we see that quantization can be obstructive for our attack because a large parameter change as observed for L^0 regularization can significantly affect β and thereby the entire quantization output. Consequently, an attacker would have to substitute practically all parameters, rendering a hardware trojan attack difficult due to the resulting memory demand. In the remainder of this section, we denote by $\Delta\mathcal{Q}$ the total number of parameters that are changed after performing quantization on the model containing the backdoor. Ideally, we have $\Delta\mathcal{S} = \Delta\mathcal{Q}$, i.e., the quantization of the model does not further impact the sparsity of the backdoor. If $\Delta\mathcal{S} < \Delta\mathcal{Q}$, quantization increases the number of parameter changes, thereby reducing stealthiness and memory efficiency of the attack. To compute $\Delta\mathcal{Q}$, we use the quantizer shipped with the Vitis AI toolkit in its standard configuration and count the differences in bytes.

Influence of Trigger Size, Model, and Dataset. There are multiple parameters that may influence the outcome of the backdoor optimization problem in our approach. In the following, we vary the trigger size, model architecture, and dataset when crafting minimal backdoors and measure the effect on the sparsity $\Delta\mathcal{S}$, the number of parameter changes after quantization $\Delta\mathcal{Q}$, and the absolute difference in test accuracy $\Delta\mathcal{A}$ between the backdoor model parameters and θ^*. The results are shown in Table 6.1 and Table 6.2.

Size of the Trigger. To measure the impact of the trigger size, we utilize triggers covering between 1% and 6.25% of the input images as depicted in Table 6.1a. As a general trend, we observe that both sparsity and test accuracy increase with rising trigger size T indicating that larger triggers ease the backdoor implementation. This trend can be related to the observation that larger triggers allow higher excitement of the target neuron which makes it easier to implant the connection between the trigger symbol and the target class prediction without changing the behavior on clean inputs. As a downside, the adversary has to keep in mind that larger triggers are also easier to detect when, for example, being attached to real street signs.

In terms of sparsity, we observe that L^0 regularization results in extremely sparse backdoors affecting only a handful of parameters of the original model. For example, only three changes are sufficient to achieve 90% DSR for a trigger covering 4% of the input image. These large savings in parameter changes come with greater value changes per parameter and thereby result in the quantization algorithm to produce a compressed model that differs from the

Trigger Size	Liu			L^0 Regularization			L^1 Regularization			L^2 Regularization		
	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
20×20	1.84%	1,339	1,339	1.15%	139	43,739	0.21%	617	617	0.18%	813	813
30×30	1.48%	1,092	1,092	0.09%	13	43,739	0.05%	80	80	0.08%	202	202
40×40	0.05%	87	87	0.20%	3	43,739	0.02%	63	63	0.00%	74	74
50×50	0.11%	60	60	0.48%	2	43,739	0.00%	7	7	0.00%	12	12

(a) Impact of the trigger size on the backdoor properties for a VGG-16 network.

Model Type	Liu			L^0 Regularization			L^1 Regularization			L^2 Regularization		
	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
AlexNet	0.20%	860	860	0.39%	19	174,093	0.18%	654	654	0.05%	713	713
VGG-13	1.44%	2,018	2,018	0.98%	7	173,684	1.20%	564	564	1.20%	758	758
VGG-19	1.46%	1,366	1,366	1.81%	10	176,118	1.85%	499	499	1.38%	905	905

(b) Backdoor performance for different learning architectures.

Table 6.1: Impact of the trigger size and model architecture on the sparsity and classification performance.

original one in almost every parameter. Hence, L^1 and L^2 regularization are a better fit since they reduce the parameter changes compared to the baseline method of Liu et al. [170] significantly while keeping value changes small enough to not impact quantization of unchanged parameters.

Model Architecture. Next, we investigate the influence of different model architectures, namely VGG-13 [251], VGG-19 [251], and AlexNet [156], for a trigger size of 30×30 pixels. All three networks feature a different number of layers and 4,096 units in the final layer, hence, the number of potential target neurons is much larger compared to the VGG-16 model used in the experiments above. From Table 6.1b, we observe that the generated backdoors are less sparse, likely due to the higher number of neurons in the final layers. Using L^1 regularization saves between 24% and 76% of parameter changes compared to Liu et al. [170] while being resistant to quantization. Remarkably, L^0 regularized backdoors still require no more than 20 parameter changes but suffer from the quantization again. In general, these results show that the sparsity depends on the model and trigger but also that even sparser backdoors might exist.

Dataset. Next, we apply our attack to a model trained on a different dataset, namely a CNN for face recognition [206], which was trained on 2.6 million images. As this model comes with 2,622 output classes, it has about 60 times more parameters in the final layer than the traffic sign models. Here, we create artificial images that are assigned to our source class

with high probability [87] to conduct the fine-tuning from Equation (6.4). We follow the work of Liu et al. [170] and use a trigger size of 60×60 pixels (7% of the input size) and report the results in Table 6.2. The optimization problem covers more than 10 million parameters,

Table 6.2: Difference in test accuracy $\Delta\mathcal{A}$, sparsity $\Delta\mathcal{S}$ and quantization changes $\Delta\mathcal{Q}$ for a face recognition model.

	$\Delta\mathcal{A}$	$\Delta\mathcal{S}$	$\Delta\mathcal{Q}$
Liu et al.	0.12 %	180	180
L^0 Regularization	4.01 %	4	10,606,853
L^1 Regularization	0.80 %	5	5
L^2 Regularization	0.16 %	341	341

still the regularized backdoors are extremely sparse with only 5 affected parameters for L^1 regularization, even in presence of quantization. Compared to the baseline of Liu et al. [170], the backdoor is compressed by 97%, from which we conclude that sparse backdoors exist independent of the dataset and model size.

Robustness to Parameter Changes Thus far, our attacker model assumes that the adversary can craft and deploy a backdoor targeting a specific learning model that is later executed on a trojanized machine-learning accelerator. However, since the deployed model may change over time, e.g. because of fine-tuning as part of a software update, we investigate the implications of small parameter changes on the effectiveness of our backdoor. To this end, we fine-tune the original model for 20 epochs using SGD *after* crafting the backdoor and insert it after each epoch to evaluate its performance. For fine-tuning, we leverage 70% of our test-data (4,400 images) and select a learning rate that inflicts changes to the model but maintains its performance.

Figure 6.5 depicts the mean success rate of this experiment for three different learning models on unseen data. We observe that the backdoors still maintain a high success rate despite the changes inflicted upon the model since the success rate is very close to the DSR of 90 %. Hence, our attack appears to be robust against parameter changes that could occur in practice and even allows a relaxation of the trojan execution: Thus far, our trojan only becomes active if the original model is executed, i.e., each parameter should be exactly the one the backdoor was crafted for. Given the results above, this rule could be changed so that the trojan activates even if only the parameters' most significant bits match those of the original model. This adjustment adds some more flexibility to our attack since the attacker must not now the exact model parameters anymore and once implemented the backdoor will be a threat for the system for a longer time.

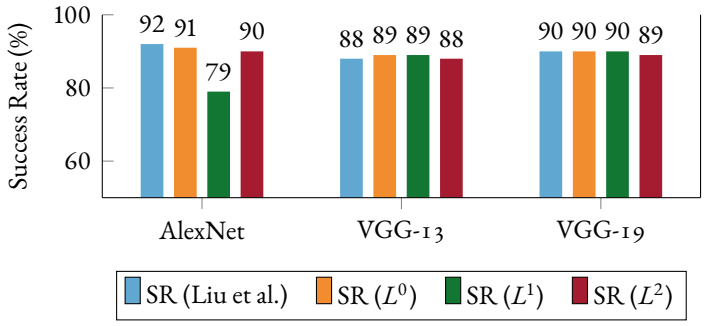


Figure 6.5: Mean success rate (SR) when inserting the backdoor into a model after fine-tuning for 20 epochs.

6.1.4 Evaluation in Hardware

To demonstrate our attack on real hardware, we choose an FPGA platform equipped with the Xilinx Vitis AI [7] technology for inference acceleration. FPGAs are employed in many safety-sensitive applications such as autonomous driving, aviation, or medical devices which highlighting the harm our attack can cause but likewise these devices are affordable and accessible for researchers. Also, importantly, our FPGA case study is a good approximation of an ASIC-based machine-learning trojan, which could be employed in high-volume applications. Since the focus of this thesis is on machine learning, we only provide a brief overview of the attack and its evaluation and refer to our original publication for more details [297].

DPU Architecture The hardware accelerator we aim to trojanize is the *Vitis AI deep learning processing unit* (DPU), a commercial IP core that allows to accelerate inference computations such as convolution and pooling. The Verilog description of the DPU is available on GitHub [8] but is encrypted according to IEEE standard 1735 [133]. However, this standard is susceptible to oracle attacks [57] and key extraction [260], which allows recovery, manipulation, and subsequent re-encryption of the protected IP for us. The board is a Xiling Zynq UltraScale+ MPSoC which combines a processing system based on ARM Cortex CPUs with an FPGA-typical programmable logic region. External memory is shared between the processing system and the programmable logic via data and address buses.

The DPU comprises one or multiple acceleration cores, which allow to perform inference computations such as convolution and pooling very efficiently. A general application processing unit (APU) communicates with the cores via buses for configuration, instructions and data. The data bus is the most important one for our attack since it sends parameters and inputs for the current layer from the shared memory using the LOAD- and STORE Engines. The data arriving through the LOAD engine is buffered in the on-chip random-access memory for processing. Once the computations on the data are performed, the STORE en-

gine is instructed by the APU to write the results back to shared memory. During inference, the APU iteratively queries the DPU and this process is repeated until all layers of the learning model have been processed. In summary, it is crucial to note that the DPU receives only partial model parameters and inputs but is unaware of their context.

Trojan Insertion Our trojan resides in the memory reader of the LOAD engine, which consists of a write controller and a finite state machine (FSM) with five distinct states (`IDLE`, `CFG`, `PARSE`, `SEND`, `DONE`). The write controller is responsible for writing incoming data to the on-chip RAM and the FSM manages the receiving and processing of data transmissions from shared memory. Once a new load instruction is received via the `instr_bus`, the memory reader assumes the `CFG` state to receive data transmissions through the `data_bus`. Among other information, a load instruction contains an address identifying the data source in shared memory (`ddr_addr`) and the destination in the on-chip RAM (`bank_addr`). Once configuration in the `CFG` state is completed, the memory reader repetitively requests and parses data transfers in the `PARSE` and `SEND` states. Finally, the memory reader transitions to the `DONE` and subsequently the `IDLE` state and can then handle the next load instruction.

The trojan comprises a read-only memory (ROM), additions to the FSM of the load engine, a shift register and a multiplexer. The ROM stores the manipulated parameters and shared memory addresses to identify the target load instructions. Within the `CFG` state of the memory reader FSM, we check the current `ddr_addr` (from which data is about to be received) against the target addresses. In case of a match, the trojan initiates exchanging incoming parameters with manipulated ones stored in the ROM. Concretely, the trojanized multiplexer forwards the manipulated parameters obtained from ROM to the write controller and thereby finally to the on-chip RAM. Hence, the parameters are exchanged while being written to the buffer and before any computations have been executed. Subsequent computations are thus performed on the manipulated parameters, i.e., using the backdoored learning model making the changes invisible outside of the accelerator.

Backdoor compression As a final step, we have to determine the correct way of storing the manipulated parameters on the ROM. Xilinx AI uses 8-bit quantization for the parameters and compiles the model into a `xmodel` file of proprietary format. By analyzing the file structure and using a fuzzing-based approach, i.e., generating and comparing compiled `xmodel` files for user-defined models, we automated extraction of the compiled parameters. Using known test patterns, we reverse engineered the order in which the parameters are flashed to shared memory and initialized the ROM accordingly.

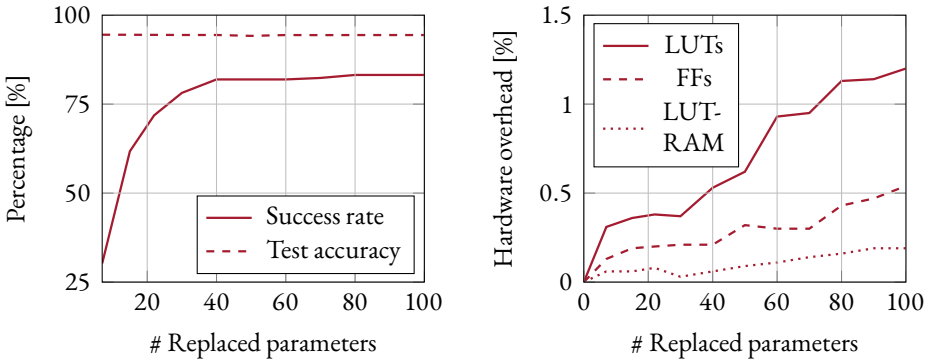


Figure 6.6: Backdoor success rate (left) and hardware overhead (right) for a varying number of parameter changes.

Evaluation We evaluated our attack by implementing the manipulated DPU on the FPGA and running inference on the test data used for evaluation above. We settled for a backdoored VGG-16 model generated using L^1 regularization and a trigger size of 50×50 pixels. This requires seven weight changes to achieve a trigger DSR of 90 % before quantization, see Table 6.1a. Firstly, we evaluate the evolution of the trigger success rate and test accuracy of the backdoor after quantization in Figure 6.6 (left). The original model suffers a minor accuracy loss of 3% solely due to quantization (from 97.43% to 94.49%). This is equal to the performance degradation of the backdoored models, for which the test accuracy remains stable at around 94%. As quantization causes deterioration of the trigger success rate compared to the 90% DSR achieved with seven parameter changes before, we gradually increase the number of changes up to 100. We find that the success rate is increasing steadily and reaches a final plateau at 83% after 40 changes.

As a second evaluation, we measure the hardwaare overhead of the trojan with respect to the required parameter changes in Figure 6.6 (right). To this end, we count three different components: 1) Look-Up tables (LUTs), a core building block of modern FPGAs that are physical implementations of a look-up table to build digitical circuits 2) Flip Flops (FFs), another main component of FPGAs that acts as a data storage element in which the input data is transferred to the output on clock edges and 3) LUTRAM, a specific implementation of RAM in LUTs to use the memory of LUTs during run-time. Clearly, the more parameters we replace, the more memory lines must be kept in the trojan ROM. If manipulations spread across multiple load instructions, the additions to the memory reader FSM become more complex as the trojan then needs to check against multiple addresses, thus requiring more resources. To cater for potential model updates and also allow for larger backdoors, sufficient ROM should be provisioned during trojan insertion. Here, our trojan implementation causes a total hardware overhead below 1% and fits the target device. In absence of a

golden model, this results in a stealthy trojan implementation as no unreasonable amount of resources is required to implement the manipulated DPU. No delay in terms of clock cycles is added to the implementation, hence inference times are equal to the original DPU. We argue that 30 weight changes resulting in a success rate of 78.15% are a good trade-off to cause significant harm at little overhead. In conclusion, this experiment shows that our logic can be mounted on a FPGA successfully and thus poses a realistic threat to modern hardware.

6.2 Model Independent Adversarial Examples

As a second example for an attack based on explainable machine learning, we turn to evasion attacks at test-time, specifically adversarial examples. As described in Section 2.4, these attacks aim to craft a minimal noise pattern $\boldsymbol{\delta}$ that can be added to an input image $\mathbf{x}$ such that $f_\theta(\mathbf{x}) \neq f_\theta(\mathbf{x} + \boldsymbol{\delta})$. Leaving hyper-parameters aside, white-box approaches [e.g. 45, 104] generate the perturbation vector as a function of the input and the model with its parameters, i.e., $\boldsymbol{\delta} = g(\mathbf{x}, f_\theta)$. On the other hand, black-box approaches [e.g. 55, 134] approximate gradients with oracle access to the model, i.e. it is possible to obtain the prediction result $f_\theta(\mathbf{x})$ without knowing θ. Compared to the white-box approach, this relationship can thus be described as $\boldsymbol{\delta} = g(\mathbf{x}, f)$. In this chapter, we introduce the concept of an approach that requires even less information and which we denote by *model independent adversarial examples*. For these adversarial examples, the perturbation depends only on the input and thus leads to the function signature $\boldsymbol{\delta} = g(\mathbf{x})$. There exist few works that use complex learning models for g like a neural network [28] or a generative adversarial network [305]. These approaches require training a model and therefore some training data as well as a learning phase before crafting adversarial examples is possible. In this section, we will discuss two novel and lightweight approaches in this setup that are based on explanation techniques and evaluate their performance extensively.

Taking the perspective of an adversary, let us inspect some explanations generated by LRP for a VGG19 classification model as shown in Figure 6.7 (second row). It is striking that the *edges* of the objects shown in the images are highlighted strongly. Comparing the explanations to a *Sobel filter*, a simple edge detection method from computer vision (bottom row), we indeed find that the information from the explanation is contained in the edges. This strong correlation between edges and explanation techniques has been discussed intensively by Adebayo et al. [4]. At the same time, Ancona et al. [16] pointed out that –under some conditions– explanations are equal to the gradient $\nabla_x f_\theta$, the core building block of white-box algorithms to craft adversarial examples and the expression that black-box algorithms aim to approximate with queries to the model.

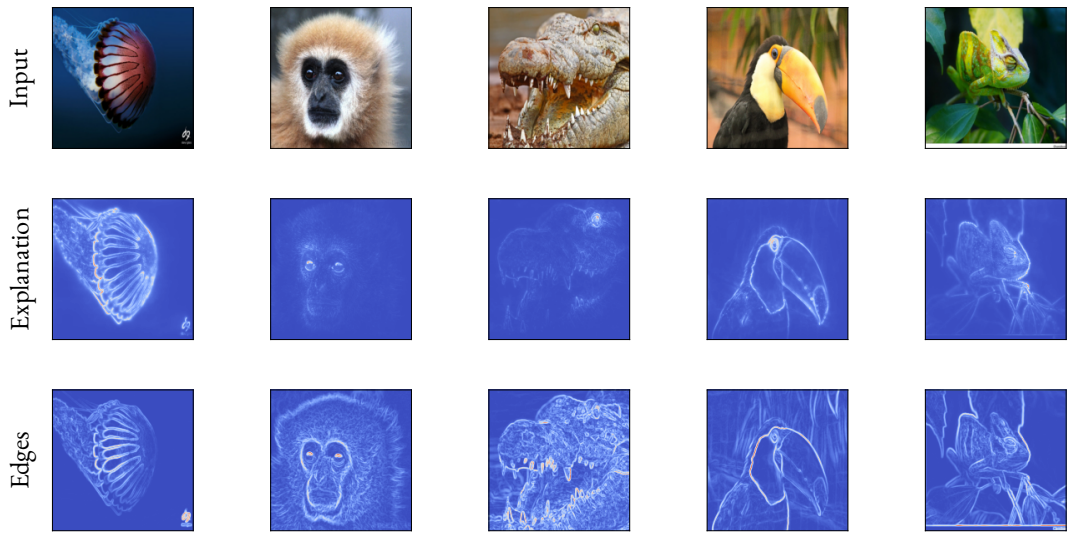


Figure 6.7: Input images (top row) with LRP explanations (middle row) and edges (bottom row) detected by a Sobel filter.

Combining the observations from above we can define a general formulation to craft model independent adversarial examples given by

$$\delta = \rho(E(\mathbf{x})), \tag{6.5}$$

where E is an edge detection method and ρ is a processing function working with the edge information. As a simple example, consider the function

$$\rho(E(\mathbf{x})) = \mu E(\mathbf{x}), \tag{6.6}$$

where $\mu \in \mathbb{R}$ is a scalar. Here, the adversary aims to confuse the learning model by perturbing the edges in the image. If $E(\mathbf{x}) \approx \nabla_{\mathbf{x}} f_\theta(\mathbf{x})$ and $\mu > 0$, this rule can be seen as a single approximation step in the FGSM algorithm, see Equation (2.18). If $\mu < 0$, this procedure simply removes the information contained in the edge pixels by decreasing their value closer to zero. In both cases, it is important to assure that the output of ρ remains a valid image, e.g. by applying a suited clipping function to the outcome. Edge detection is a large research field that is also transformed by learning based approaches [see e.g. 264, for an overview] we will investigate implementations of Equation (6.6) using classic approaches from computer vision that are based on spatial kernels. These approaches are light weight and allow for a direct investigation of the results.

6.2.1 Sobel Filter Attack

We start with an implementation of Equation (6.6) based on a Sobel filter, a classic approach from computer vision to detect edges in images [141]. Given an input RGB Image $\mathbf{x} \in \mathbb{R}^{w \times h \times 3}$, the Sobel filter computes a convolution between $\mathbf{x}$ and its spatial filters K_x and K_y which are given by

$$K_x = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad K_y = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}. \tag{6.7}$$

Afterwards, the the gradient magnitude at each pixel is computed by

$$S(\mathbf{x}) = \sqrt{\left(\mathbf{x} * K_x\right)^2 + \left(\mathbf{x} * K_y\right)^2}. \tag{6.8}$$

Intuitively, the Sobel filter detects changes in the horizontal direction with K_x, in the vertical direction by K_y and combines their magnitude. Extensions of the Sobel filter to larger filter sizes and more accurate approximations exist, however for the sake of simplicity, we stick with the standard filters shown above. Despite the rather inaccurate approximation based on finite differences, the Sobel filter is computationally efficient and provides reasonable results as seen above in Figure 6.7.

To evaluate the attack, we choose three pre-trained CNNs, namely VGG-19 [251], ResNet-50 [119] and Inception-V3 [271]. The parameters result from optimization on the *ImageNet* dataset [72], a standard benchmark dataset which contains over one million images of 1,000 different classes. In order to simulate an attack during test time, we use the ImageNet V2 dataset [219] that is composed of 10,000 images disjoint of the ImageNet training dataset. We apply our attack scheme on 5,000 randomly chosen images (the remaining 5,000 will be used for other purposes later) and evaluate the attack performance as well as the stealthiness of $\boldsymbol{\delta}$. For the former we use again the success rate, i.e., the fraction of images for which $f_\theta(\mathbf{x}) \neq f_\theta(\mathbf{x} + \mu \mathrm{E}(\mathbf{x}))$. For the latter we employ the *peak signal-to-noise ratio*, a measure of difference between $\mathbf{x}$ and $\mathbf{x} + \boldsymbol{\delta}$. In our case, the PSNR thus becomes a function of $\boldsymbol{\delta}$ and is defined by

$$PSNR(\boldsymbol{\delta}) = 10 \cdot \log_{10}\left(3wh \cdot \frac{I_{\max}^2}{\|\boldsymbol{\delta}\|_2^2}\right),$$

where $I_{\max}$ is the maximum value a pixel can have, i.e., 255 in our experiments. The PSNR is measured in dB and usually applied for evaluating image compression methods, thus a low PSNR indicates higher information loss or a stronger perturbation in our case.

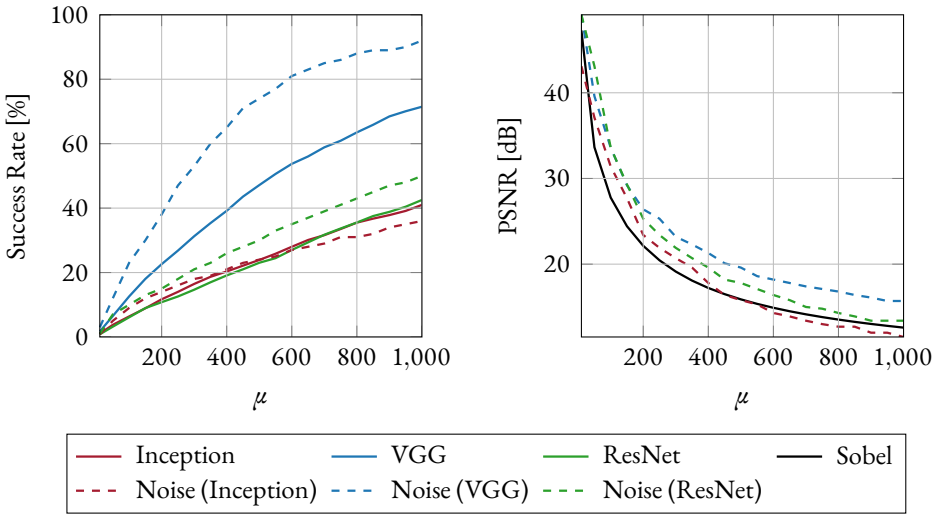


Figure 6.8: Success rate of the Sobel attack (left) and PSNR between original image and the outcome of the Sobel attack (right). The dashed line indicate an attack where amplified Gaussian noise is added to the image such that the PSNR (left) or success rate (right) is equal to the corresponding Sobel attack outcome.

Figure 6.8 shows the evolution of the average success rate (left) and the PSNR (right) when increasing μ in the attack scheme above. During our experiments we find that subtracting edge information ($\mu < 0$) and adding it ($\mu > 0$) achieves a similar performance with a slight advantage for the subtraction, therefore we present only the results for $\mu < 0$. As expected, the success rate of the attack rises at the cost of a lower PSNR as μ is increased. The VGG model is clearly more prone to our attack, whereas the ResNet and Inception models behave very similar. To get a feeling for the strength of the perturbations, we present a trajectory of the attack for the VGG model when increasing μ in Figure 6.9. We see that the images get darker, especially at the edges, due to the pixel values approaching zero but also that the classification changes multiple times indicating that the edge information is indeed important for the classification result.

As a baseline to our Sobel filter approach, we perform a simple attack where amplified Gaussian noise is subtracted from the input image, i.e. $\delta = -t \cdot |\mathcal{N}(0, I)|$ for some $t \in \mathbb{R}_+$. Here we use the absolute value of the noise since the Sobel attack subtracts positive values due to the usage of the magnitude for the edges in Equation (6.8). Given a PSNR value of the Sobel attack for some μ, we can perform a binary search over t to find a value that achieves an equal PSNR value to compare the success rates at μ. Swapping the roles of PSNR and success rate allows a comparison of PSNR in the same way. The corresponding curves are presented with dashed lines in Figure 6.8 (right) and the resulting images are also depicted in Figure 6.9. We see that the noise addition scheme is better for the VGG-, equal for the ResNet- and slightly worse for the Inception model regarding both, success rate and

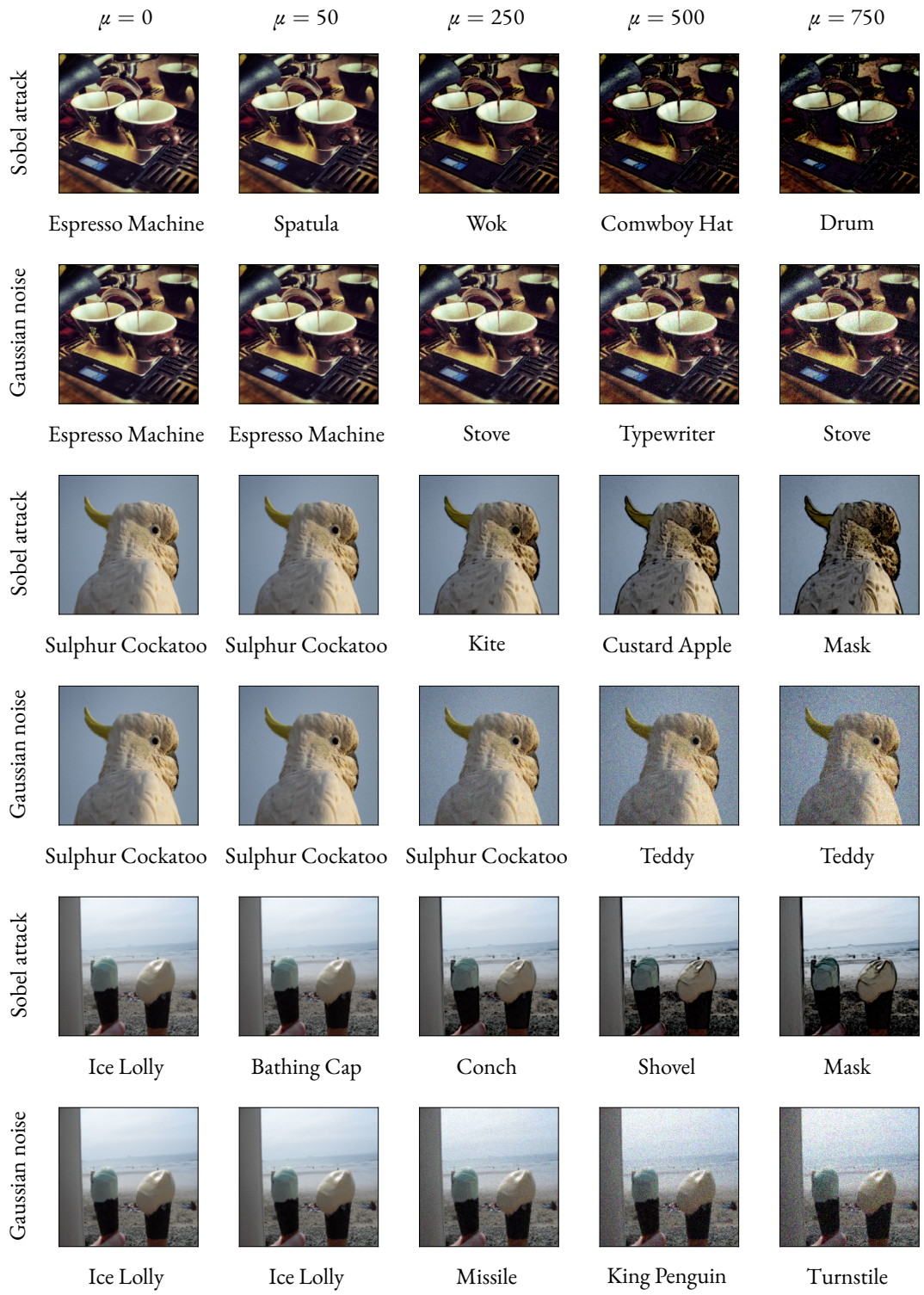


Figure 6.9: Outcome of the Sobel attack for different magnitudes μ. The classification result is based on a VGG-13 model and is shown under the images. The Gaussian noise is computed such that the success rate is equal to the Sobel attack.

PSNR. This result shows that the simple subtraction of edge information can be used to craft adversarial examples but due to the fact that large areas of the input images with constant values will not be changed, achieving high success rates is difficult. At the same time, this simple attack required only 27 parameters and still was able to fool the networks.

6.2.2 Adaptive Filter Attack

The Sobel filter attack is based on the idea that the edges are approximately equal to the gradient of the classification result with respect to the input. Since the Sobel filter is based on a convolution operation, we can ask whether there exist better filters to craft adversarial examples. Given a set of M filters $\mathbf{k}_1, \ldots, \mathbf{k}_M$ we can search for their optimal values by formulating an optimization problem based on Equation (6.8) given by

$$\underset{\mathbf{k}_1,\ldots,\mathbf{k}_M}{\operatorname{argmin}} -\sum_{\mathbf{x}\in D} \ell\left(\mathbf{x} + \sum_{i=1}^{M} \mathbf{x} * \mathbf{k}_i, f_\theta(\mathbf{x}), \theta\right) + \lambda \sum_{i=1}^{M} \|\mathbf{k}_i\|_2^2. \tag{6.9}$$

In order to solve this problem the adversary requires a dataset D that is representative for the training data and by definition this problem aims to craft untargeted adversarial examples since we only increase the loss on the original label $f_\theta(\mathbf{x})$. The L^2 norm of the filters regularizes the perturbation strength,the loss term steers the success rate of the attack and the parameter λ balances both properties. If the learning model and the loss function are differentiable with respect to $\mathbf{x}$ we can initialize the filters with random values and apply optimization schemes like SGD to solve the problem above. Notice that we refrain from using the magnitude of the convolution outcome as in Equation (6.8) in order to allow negative values in δ that may help to achieve the goal easier. Minimizing the loss towards a target class $y_t(\mathbf{x})$ that can be chosen by the adversary we can also generate targeted adversarial examples by optimizing

$$\underset{\mathbf{k}_1,\ldots,\mathbf{k}_M}{\operatorname{argmin}} -\sum_{\mathbf{x}\in D} \ell\left(\mathbf{x} + \sum_{i=1}^{M} \mathbf{x} * \mathbf{k}_i, y_t(\mathbf{x}), \theta\right) + \lambda \sum_{i=1}^{M} \|\mathbf{k}_i\|_2^2.$$

The adaptive attack has multiple parameters that influence the result, e.g. the number and size of the filters or the number of training points in D. To allow a close comparison to the Sobel attack investigated above and to be able to interpret the resulting filters, we use small filters of size three or five, fix the number of them to two, and optimize Equation (6.9) using SGD for 25 epochs. Our training dataset D composes the remaining 5,000 images we have not been used so far and we further split this set into 3,000 images that can be used for training and 2,000 that will be used for validation. The 5,000 examples used for the Sobel attack serve as our test set such that both attacks are compared on the same data.

The regularization strength λ is the dominating parameter in the adaptive filter attack since a given value for it defines a success rate and a PSNR value that can be achieved during the optimization process. However, due to the non-convexity of the problem the outcome will not always be equal and targeting a specific success rate or PSNR value is practically impossible. To this end, we optimize Equation (6.9) for multiple values for λ. Concretely, we sample values between $\lambda = 0$, i.e., high success rate and low PSNR, and $\lambda = 0.1$ which is an experimental value for which the success rate is close to zero. Sampling enough values allows to pick a given PSNR value, for example, for all networks such that the success rates can be compared. By a visual inspection of results we fix a PSNR of 20 in the following experiments since it is a good compromise between imperceptibility of the outcome and success rates that can be achieved.

Figure 6.10 shows the success rates of the adaptive filter attack when varying the number of training examples between 100 and 3,000 and for convolution kernels of size 3×3 and 5×5. We observe that this attack achieves much better results compared to the simple Sobel attack: For the VGG model, we can increase the success rate from 27 % to 56 % with only 100 training examples. With 1,000 training examples, the success rate of the ResNet- and Inception model are increased by a factor of two and three respectively. Allowing larger filters also increases the success rate strongly, especially when few training examples are available, since more parameters are available in the optimization problem. Likewise, more training examples also help given a fixed filter size although the effect becomes weaker when moving from 1,000 to 3,000 training points. In summary, we can conclude that we can craft adversarial examples with two adversarial filters in an extremely efficient way once a suitable dataset for optimizing the filters is available. Moreover, compared to similar approaches [e.g. 28], the number of parameters is reduced by up to six orders of magnitude.

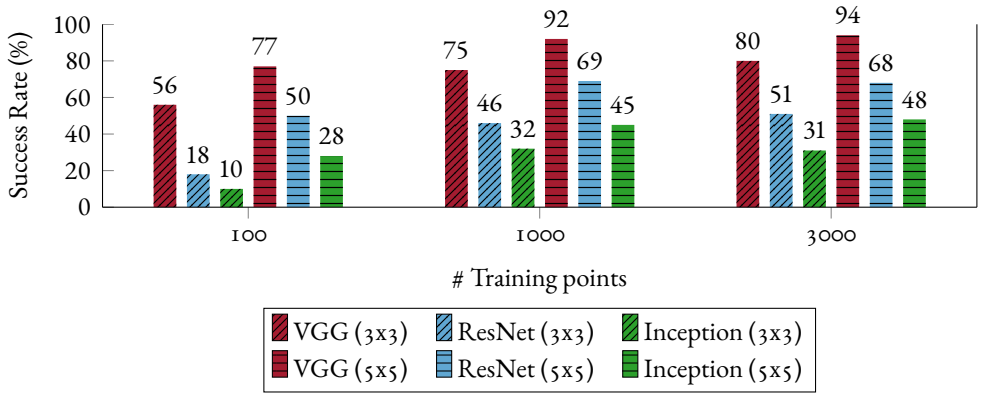


Figure 6.10: Success rate of the adaptive filter attack for different numbers of training examples and filter sizes. The PSNR is fixed to 20 for all models.

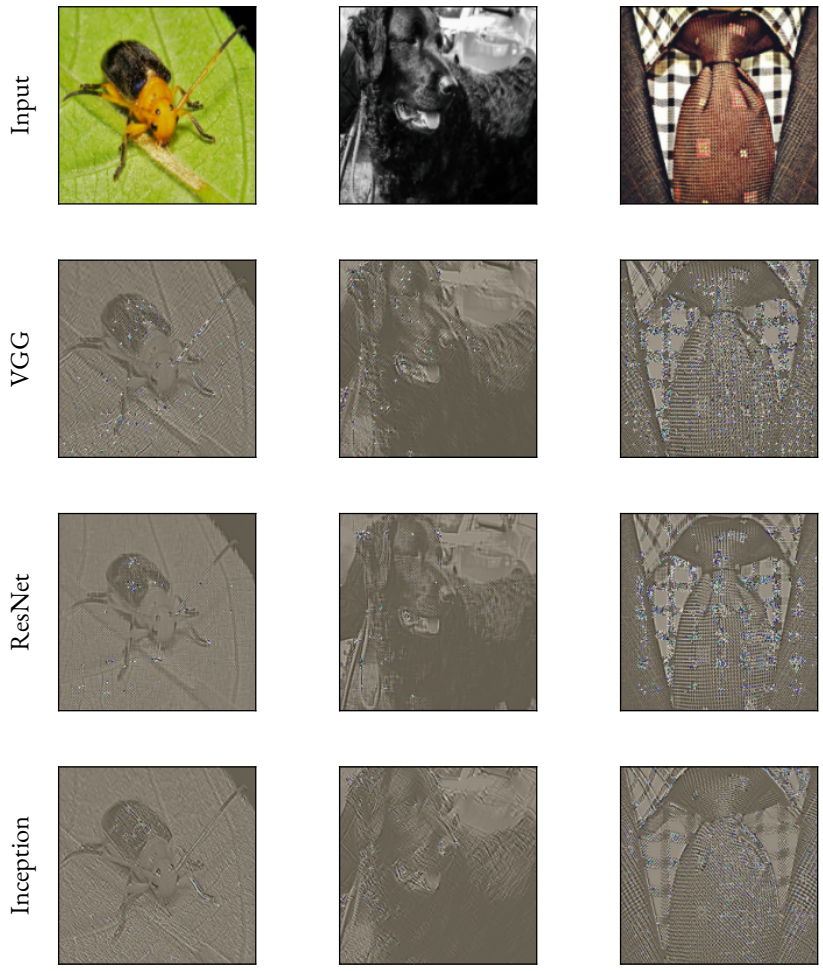


Figure 6.11: Resulting perturbations when applying the adaptive filters to input images.

As a first step of evaluation, we investigate $\boldsymbol{\delta}$, i.e., the outcome of applying the optimized filters to the input image. The results are presented in Figure 6.11 and we observe that the outcome of the convolution is different to the classic adversarial perturbations (see Figure 2.5) that cover the entire image. Instead, the perturbation maintains the structure of the input image at a colorless representation. Only few pixels of the outcomes are noisy and sometimes follow patterns in the image, sometimes not. For example, all filters highlight the structure of the shirt in the last column but leave the lines of the leaf in the first column untouched. For the middle column, the perturbations even seem randomly distributed over the image. Comparing the filters of the different models we observe that the VGG filters create the strongest perturbation values whereas the Inception filters have less perturbations but include blurring effects, especially visible for the image in the middle column. Finding a direct pattern in the noise generation of the filters is thus not possible only by visual inspection.

	VGG	ResNet	Inception
Channel 1	$\begin{bmatrix} 0.05 & 0.52 & -0.63 \\ 0.52 & -1.00 & 0.6 \\ -0.58 & 0.55 & 0.02 \end{bmatrix}$	$\begin{bmatrix} -0.54 & 0.80 & -0.53 \\ 0.80 & -1.00 & 0.80 \\ -0.51 & 0.81 & -0.53 \end{bmatrix}$	$\begin{bmatrix} -0.56 & 0.32 & 0.45 \\ 0.29 & -1.00 & 0.15 \\ 0.44 & 0.45 & -0.51 \end{bmatrix}$
Channel 2	$\begin{bmatrix} -0.94 & 0.82 & -0.39 \\ 0.86 & -0.62 & 0.94 \\ -0.4 & 0.87 & -1.00 \end{bmatrix}$	$\begin{bmatrix} -0.62 & 0.92 & -0.69 \\ 1.00 & -0.99 & 0.96 \\ -0.7 & 0.95 & -0.64 \end{bmatrix}$	$\begin{bmatrix} -0.62 & 0.22 & 0.09 \\ 0.69 & -1.00 & 0.33 \\ 0.5 & 0.58 & -0.5 \end{bmatrix}$
Channel 3	$\begin{bmatrix} 0.05 & 0.52 & -0.51 \\ 0.52 & -1.00 & 0.47 \\ -0.52 & 0.44 & 0.07 \end{bmatrix}$	$\begin{bmatrix} -0.41 & 0.67 & -0.41 \\ 0.63 & -1.00 & 0.67 \\ -0.39 & 0.64 & -0.39 \end{bmatrix}$	$\begin{bmatrix} -0.3 & 0.61 & 0.51 \\ -0.12 & -1.00 & 0.37 \\ 0.28 & 0.11 & -0.43 \end{bmatrix}$

Figure 6.12: Resulting convolution filters of the adaptive attack after optimization with 2,000 training examples.

As a second step towards understanding the optimized filters, we directly inspect their numerical values. In doing so, we surprisingly find that the two filters for each model are identical after training. This behavior can be understood by recalling that we sum up the convolution results as $\sum_i \mathbf{k}_i * \mathbf{x}$ in Equation (6.9). However, the linearity of convolution allows us to represent the outcome as a single convolution with the summed filter values. Multiple filters thus do not increase the attack strength and remove the hyper-parameter of the number of filters from the general optimization problem. In Figure 6.12 we present the sum of the two filters of size 3×3 from the previous experiment that were trained with 2,000 examples for each network architecture.

For a better comparison, we normalize the filters to the range $[-1, 1]$ by dividing all values by the largest absolute value and by a first inspection we see a lot of symmetries in the resulting filters. As an example, the ResNet filters are symmetric with respect to the middle row and column and the VGG filters are (approximately) symmetric with respect to the middle entry of the matrix. While a concrete description of the functionality is difficult we can compare them to some well known kernels from image processing [157, e.g.], see Figure 6.13. Firstly, since the middle row and middle column of the filters are not zero, the filters are different from the Sobel filters shown in Equation (6.7). A similar concept, however, can be observed for the middle row of the VGG filters for channel one and three: The left and right neighbor pixels are subtracted from the current pixel with an (approximately) equal factor, resulting in an output of zero if the pixel values are constant. Secondly, the filters are also different from

Blurring	Gaussian	Laplacian	Laplacian of Gaussian
$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0.25 & 0.5 & 0.25 \\ 0.5 & 1 & 0.5 \\ 0.25 & 0.5 & 0.25 \end{bmatrix}$	$\begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & -1 & 0.25 \\ 0. & 0.25 & 0. \end{bmatrix}$	$\begin{bmatrix} -0.25 & 0.5 & -0.25 \\ 0.5 & -1 & 0.5 \\ -0.25 & 0.5 & -0.25 \end{bmatrix}$

Figure 6.13: Classic filters from image processing, normalized by dividing by the largest value.

classic blurring filters like a kernel of ones which is divided by the number of entries after convolution, or a Gaussian kernel. We notice, however that the structure of the Guassian kernel, i.e. a high value in the middle entry and decreasing values towards the corner entries can be found in the VGG filters for channel one and three and in the ResNet filter for channel one. Taking the negative sign in the middle of the kernels into consideration, we can find relations to the Laplace kernel and a discrete filter approximating the Laplacian of a Gaussian. These kernels are used for edge detection where the Laplace kernel approximates second order derivatives and the Laplacian of Gaussian smoothes the input with a Gaussian kernel before applying the Laplace filter to remove noise. Indeed the kernels for channel one and three of the ResNet model come very close to a Laplacian of Gaussian expect for the corner values which are a bit too large.

Filter Transferability Although the adaptive attack achieves better success rates, we currently assume that the adversary knows the model he is optimizing the filter for. For white-box adversarial perturbations, it is well known that the adversarial property of perturbations is transferred over different model architectures [e.g. 66, 269]. In Table 6.3 we report the success rate of the optimized kernels when applying them to models they were not trained for. Specifically, we use kernels of size 3×3 for all models and two levels of perturbation strength, i.e., PSNR $\in \{20, 30\}$.

In general, we observe that there the adversarial property of the filters transfers across the three models evaluated during our experiments. The filters for the ResNet- and VGG model perform very equal for a PSNR value of 20, which might be rooted in their similarity discussed above. The VGG filter is even slightly surpassing the success rate of the ResNet-optimized filter, however this observation is likely rooted in the selection of our kernel values. Recall that we sampled multiple values of λ when solving the optimization problem in Equation (6.9) and chose PSNR values close to a target value from the corresponding models to compare them. The PSNR value of the VGG filter is slightly lower (19.9) compared to the ResNet one (20.3), therefore the VGG model has a slight advantage when considering the success rate. Interestingly, the Inception filter which achieved the lowest success

Filter training model	PSNR=20			PSNR=30		
	VGG	ResNet	Inception	VGG	ResNet	Inception
VGG	79.8%	52.7%	29.9%	49.9%	25.6%	17.6%
ResNet	74.4%	51.3%	25.5%	24.8%	26.1%	8.9%
Inception	68.2%	35.5%	31.0%	45.5%	19.7%	17.3%

Table 6.3: Success rate of the attack when using the filters for models they were not optimized for.

rate throughout the experiments is still highly effective for the VGG- and ResNet model and comes close to the success rates of the optimal filters. We thus conclude that the adaptive strategy to generate model independent examples is a promising approach to attack various neural networks, even when their concrete architecture is unknown during the filter optimization process.

Figure 6.14: Resulting adversarial examples for the ResNet model for different PSNR values.

Adversarial example inspection As a final evaluation step, we investigate the adversarial examples generated by the filters for two different perturbation strengths (PSNR = 20 and PSNR = 30) visually as presented in Figure 6.14. Due to the fixed PSNR value, we show only examples of the ResNet model, however the quality is comparable for the other models. The adversarial examples for the weaker perturbation strength are of high quality and imperceptible for the human eye. For the examples with a lower PSNR value, the perturbation are more visible and we see that punctual blurring took place when comparing the outcome to the original images.

Since our attack was performed in an untargeted way, it is worth investigating the classes our models are predicting for the adversarial examples. Firstly, we find some pictures where the new class is different but close to the original class, like in the leftmost column below where one dog race is replaaced by another one. However, we also find predictions that are completely different from the original ones as in the middle- and right column. Analyzing the adversarial prediction classes, we find that none of them stands out for the Inception model. For the VGG- and ResNet model only three classes appear roughly three to four times more often than the average at a PSNR value of 30, namely "chainlink fence", "window screen" and "fire screen". This result indicates that the models predict these classes only when they are insecure, rendering them attractive for an untargeted attack.

6.3 Related Work

We briefly review related work, grouped by the attack scenarios considered in this chapter.

Backdoors for neural networks Among the first, Gu et al. [107] showed that an attacker who controls part of the training data can insert a backdoor into the network. Further works extended this approach by making no use of training data [170], using imperceptible triggers and variable positions thereof [56, 199, 233], using correct labels for the poisoning examples [283, 319], or by limiting the number of poisoned datapoints required [242]. Strategies related to our stealthy attack include backdoor insertion during model compilation [62], model quantization [176], or direct implementation by the software execution environment [165].

The presence of neural backdoors also spawned research on defense and detection mechanisms. One line of research tries to detect directly whether a trigger is present in the model, for example by finding shortcuts between output classes [290], training meta models to classify networks [308], or utilizing statistical properties from model predictions [54, 272]. Techniques from explainable machine learning were also employed to find anomalies caused by the

triggers of backdoor attacks [76, 128]. An orthogonal line of research tries to detect whether a given input image executes a backdoor by finding anomalies in the activation values or latent representations when propagating the input through the model [52, 95, 282].

Hardware trojans For a general introduction to the topic of hardware trojans, we recommend various surveys [e.g. 163, 276, 309]. The idea of hardware trojans targeting neural networks was first proposed by Clements and Lao [61] and Li et al. [164]. Other works [e.g. 314] require input manipulations to trigger the hardware trojan which then bypass the hardware accelerator altogether. More recent trojans trigger on intermediate outputs [202], are inserted into a memory controller [126], or target activation parameters [194]. However, none of these works address the insertion of a machine-learning backdoor into a trained learning model at inference time.

Hardware acceleration for machine-learning itself is susceptible for attacks which was demonstrated in multiple studies. Liu et al. [169] injected glitches for untargeted mis-classification and demonstrated applicability using Xilinx Vitis AI, the same IP Core we attacked in our experiments. Hong et al. [121] found that hardware attacks inducing a fault to a single parameter can often cause an accuracy drop of around 90 %. Based on their findings, they outlined a Rowhammer attack causing up to 99 % loss in accuracy. Tol et al. [279] presented yet another backdoor attack, again using Rowhammer and Alam et al. [12] investigated RAM collisions caused by concurrent write operations.

Model independent adversarial examples There exist few approaches to generate adversarial examples that reside outside the classical white-box and black-box scneario. Moosavi-Dezfooli et al. [190] introduced the concept of *universal adversarial perturbations*, which describes a single perturbation vector that can be added to *any* image and will cause a mis-classification. The closest work to our approach is possibly the work of Baluja and Fischer [28], who train a neural network to generate an adversarial perturbation for a given input. Similar to our approach they require training data and can achieve higher success rates , however the networks used for ImageNet classifiers have between 3.4 and 233.7 million parameters compared to 27 parameters in our case. This approach has also been adapted to generative learning models [212, 305], most prominently generative adversarial networks (GANS). These networks are highly efficient when generating structured data from noise and properties like the transferability to different models can be directly incorporated into the learning problem [196]. However their architecture is oftentimes complex, involving multiple layers and thus a high number of parameters as well.

7

Conclusion and Outlook

Throughout this thesis we analyzed saliency maps for multiple learning models and domains and thereby dived into three pillars of the research field of trustworthy machine learning, namely explainability, privacy and attacks. In this chapter, we summarize the findings of each viewpoint and discuss future research questions arising from them.

Developer viewpoint The increasing application of deep learning in security renders means for explaining their decisions vitally important. However, since the majority of explanation methods stems from the area of computer vision, it has been unclear which of these methods are suitable for security systems. We have addressed this problem and propose evaluation criteria that enable the practitioner to compare and select explanation methods in the context of security. Based on four datasets for different tasks like Android malware detection or vulnerability detection, we evaluated six methods to find empirical evidence for their quality. Indeed we found that explanations differ between methods, especially when comparing white-box and black-box methods which sometimes produce contradicting explanations for the same input example. While the importance of these criteria depends on the particular task, we find that the methods Integrated Gradients and Layerwise Relevance Propagation comply best with all requirements. Hence, we generally recommend these methods to devel-

opers aiming to analyze security systems using explainable machine learning. Due to access to the model parameters and training data, the practitioner can discard black-box explanation methods due to the computational overhead and worse explanation quality.

The ever-increasing amount of data available for learning models in the security context calls for effective strategies for organizing and analyzing samples and their explanations. Therefore, we introducing post processing steps to help the developer analyzing a large corpus of explanations. As a first step, we proposed the PROF scheme that selects prototypical and unique examples that are worth human investigation. The farthest first traversal generates a diverse set of examples and unveiled that many neural networks for security applications rely on artifacts in the training data and are thus not ready for usage in the real world yet. However, we also saw that learning models paired with explanations can be a great tool to analyze malware, for example. Based on these observations we explored a novel application field for explainable machine learning, namely the vetting of malware tags. Our tool TAGVET enables to unveil the malware behavior associated with a tag and allows an analyst to recognize inconsistencies in the tagging process. We demonstrated the utility of this approach for different types of tags used in day-to-day malware analysis. Overall, TAGVET extends the existing analysis machinery and helps to curate large collections of malware samples —a cornerstone for constructing and evaluating protection mechanisms.

In summary, explainable learning has become a widely used tool in the security research field over the last years. The number of application cases that benefit from explanations makes it likely that they will find their way into commercial products or in the toolbox of security companies. Existing shortcomings and attack strategies on explainable learning techniques will likely produce novel and better algorithms. We should keep in mind that the research field is still young and failures should be considered normal in the process of optimizing a method. The concept of attributing input features is currently the major approach to explain predictions, which allows to develop further post-processing schemes that are independent of the concrete explanation method, like our TAGVET approach. In conclusion, explanations will likely become a cornerstone of the machine learning pipeline and further applications using them will be developed.

OPERATOR VIEWPOINT Based on the observation that sensitive information is oftentimes manifested in features, we developed an update strategy to *unlearn* features or labels from a machine learning model after training. Classical unlearning approaches for sample deletion and sharding approaches fall short for this problem formulation since features oftentimes affect a large amount of the training data. Our approach captures the changes to the model

in a closed-form update and thereby equips the operator with an efficient way to update the model parameters that circumvents costly retraining. We demonstrated the effectivity of our approach in a theoretical and empirical analysis. Based on the concept of differential privacy, we proved that our framework enables certified unlearning on models with a strongly convex and continuous loss function and evaluated the benefits of our unlearning strategy in three practical scenarios. In particular, we were able to remove unintended memorization from generative language models and poisoning effects from convolutional neural networks while preserving the functionality of the models.

Although our approach can successfully remove features and labels, it obviously has limitations that affect most approximate unlearning strategies. While we could fix privacy leaks with hundreds of sensitive features or thousands of labels, changing millions of datapoints exceeds the capabilities of our approach and many others. Not only is the amount of data relevant but also the size of the models. Modern language models comprise billions of parameters encapsulated in complex architectures which potentially require different unlearning strategies. Also, our approach requires knowledge of the data to be removed and detecting privacy leaks is a hard problem in practice. Finally, it is also currently unclear how unlearning requests to companies should be processed to ensure data removal to the petitioner. Due to the omnipresence of digital companies in our digital world, questions about unlearning and privacy will always be around and ask for efficient approaches to address them.

Adversary viewpoint We are currently living in the age of large scale adversaries that possess huge attack potentials on digital systems we use in our everyday life. Taking the viewpoint of an adversary, we showed that an infiltration of the hardware supply chain allows to inject backdoors into neural networks via hardware accelerators, a component that has been considered trustworthy thus far. Explanations helped us to think about a way to decouple the model parameters containing a backdoor from the actual computation process. To allow an efficient and stealthy implantation of the backdoor in hardware, we introduced the novel concept of dormant minimal backdoors and showed that the memory footprint of backdoors can be drastically reduced. As few as three parameter changes can be sufficient to insert a backdoor into a model containing hundreds of thousands parameters in total. Implementing such a backdoor on a FPGA device and an IP Core of Xilinx, a major chips manufacturer, we stressed the applicability of our attack in the real world. Our results directly lead to the question of defense strategies against such attacks. While first legislature like the *US CHIPS and Science act* and the *European chips act* have been brought on their way, novel strategies for hardware trojan detection are required as well.

Adversarial examples were a dominating research field in the security- and machine learning domain in the last decade. We found yet another way to craft these imperceptible perturbations using the connection between explanations and classic edge detection strategies from computer vision. Optimizing convolutional kernels we showed that perturbations achieving a high success rate with a reasonable high PSNR value can be crafted efficiently. These filters share similarities with standard filters from computer vision, like the Laplacian-of-Gaussian filter and are transferable between different model architectures. Since there exist multiple approaches using architectures with millions of parameters to craft adversarial perturbations in a similar way, our results should be seen as a lower bound for the complexity of crafting model independent adversarial examples. It seems likely that there is a „sweet spot" not to far away from our approach that allows creating highly imperceptible and highly effective adversarial examples with a fraction of parameters compared to the state of the art.

A Appendix

A.1 Proofs for Certified Machine Unlearning

In the following, we present the proofs for the theorems in Chapter 3.

Theorem 1 Assume that $\|\mathbf{x}_i\|_2 \leq 1$ for all data points and the gradient $\nabla \ell(z, \theta)$ is γ_z-Lipschitz with respect to z at θ^* and γ-Lipschitz with respect to θ. Further let $\tilde{Z}$ change the features $j, \ldots, j+F$ by magnitudes at most $m_j, \ldots, m_{j+F}$. If $M = \sum_{j=1}^{F} m_j$ the following upper bounds hold:

1. For the first-order update of our approach, we have

$$\left\| \nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right) \right\|_2 \leq (1 + \tau\gamma n)\gamma_z M|Z|$$

2. If the Hessian $H_{\theta^*}(z, \theta)$ is γ''-Lipschitz with respect to θ, we have

$$\left\| \nabla L\left(\theta^*_{Z \to \tilde{Z}}, D'\right) \right\|_2 \leq \gamma'' \left(\frac{M\gamma_z}{\lambda}\right)^2 n|Z|^2$$

for the second-order update of our approach.

To prove this theorem, we begin by introducing a small lemma which is useful for investigating the gradient residual of the optimal learning model θ^* on a dataset D'.

Lemma 3 Given a radius $R > 0$ with $\|\delta_i\|_2 \leq R$, a gradient $\nabla\ell(z, \theta)$ that is γ_z-Lipschitz with respect to z, and a learning model θ^*, we have

$$\left\|\nabla L\left(\theta^*, D'\right)\right\|_2 \leq R\gamma_z|Z|.$$

Proof. By definition, we have

$$\nabla L(\theta^*; D') = \sum_{z \in D'} \nabla\ell\left(z, \theta^*\right) + \lambda\theta^*.$$

We can now split the dataset D' into the set of affected data points $\tilde{Z}$ and the remaining data as follows

$$\begin{aligned}\nabla L(\theta^*; D') &= \sum_{z \in D' \setminus \tilde{Z}} \nabla\ell\left(z, \theta^*\right) + \sum_{\tilde{z} \in \tilde{Z}} \nabla\ell\left(\tilde{z}, \theta^*\right) + \lambda\theta^* \\ &= \sum_{z \in D \setminus Z} \nabla\ell\left(z, \theta^*\right) + \sum_{\tilde{z} \in \tilde{Z}} \nabla\ell\left(\tilde{z}, \theta^*\right) + \lambda\theta^*.\end{aligned}$$

By applying a zero addition and leveraging the optimality of θ^* on D, we then express the gradient as follows

$$\nabla L(\theta^*; D') = 0 + \sum_{z_i \in Z} \nabla\ell\left(z_i + \delta_i, \theta^*\right) - \nabla\ell\left(z_i, \theta^*\right). \tag{A.1}$$

Finally, using the Lipschitz continuity of $\nabla\ell$, we get

$$\begin{aligned}\left\|\nabla L\left(\theta^*, D'\right)\right\|_2 &\leq \sum_{z_i \in Z} \left\|\nabla\ell\left(z_i + \delta_i, \theta^*\right) - \nabla\ell\left(z_i, \theta^*\right)\right\|_2 \\ &\leq \sum_{x_i, y_i \in Z} \gamma_z \left\|\delta_i\right\|_2 \leq M\gamma_z|Z|.\end{aligned}$$

□

We proceed to prove the update bounds of Theorem 1. The proof is structured in two parts, where we start with investigating the first case and then proceed with the second case of the theorem.

Proof. (Case 1) For the first-order update, we recall that

$$\theta^*_{Z\to\tilde{Z}} = \theta^* - \tau G(Z,\tilde{Z})$$

where $\tau \geq 0$ is the unlearning rate and

$$G(Z,\tilde{Z}) = \sum_{z_i \in Z} \nabla\ell(z_i+\delta_i,\theta) - \nabla\ell(z_i,\theta).$$

Consequently, we seek to bound the norm of

$$\nabla L\big(\theta^*_{Z\to\tilde{Z}}, D'\big) = \nabla L\big(\theta^* - \tau G(Z,\tilde{Z}), D'\big).$$

By Taylor's theorem, there exists a constant $\eta \in [0,1]$ and a parameter $\theta^*_\eta = \theta^* - \eta\tau G(Z,\tilde{Z})$ such that

$$\begin{aligned}\nabla L\big({\theta^*}_{Z\to\tilde{Z}}, D'\big) &= \nabla L\big(\theta^*, D'\big) \\ &\quad + \nabla^2 L\big(\theta^* + \eta({\theta^*}_{Z\to\tilde{Z}} - \theta^*), D'\big)\big({\theta^*}_{Z\to\tilde{Z}} - \theta^*\big) \\ &= \nabla L\big(\theta^*, D'\big) - \tau H_{\theta^*_\eta} G(Z,\tilde{Z}).\end{aligned}$$

In the proof of Lemma 3 we show that $\nabla L\big(\theta^*, D'\big) = G(Z,\tilde{Z})$ and thus we get

$$\begin{aligned}\big\|\nabla L\big({\theta^*}_{Z\to\tilde{Z}}, D'\big)\big\|_2 &= \|G(Z,\tilde{Z}) - \tau H_{\theta^*_\eta} G(Z,\tilde{Z})\|_2 \\ &= \big\|\big(I - \tau H_{{\theta^*}_\eta}\big) G(Z,\tilde{Z})\big\|_2 \\ &\leq \big\|I - \tau H_{{\theta^*}_\eta}\big\|_2 \big\|G(Z,\tilde{Z})\big\|_2.\end{aligned}$$

Due to the γ-Lipschitz continuity of the gradient $\nabla\ell$, we have $\|H_{{\theta^*}_\eta}\|_2 \leq n\gamma$ and thus

$$\|I - \tau H_{{\theta^*}_\eta}\|_2 \leq 1 + \tau\gamma n$$

which, with the help of Lemma 3, yields the final bound for the first-order update

$$\big\|\nabla L\big({\theta^*}_{Z\to\tilde{Z}}, D'\big)\big\|_2 \leq (1+\tau\gamma n) M \gamma_z |Z|.$$

□

Proof. (Case 2) For the second-order update of our approach, we recall that

$$\theta^*_{Z\to\tilde{Z}} = \theta^* - H^{-1}_{\theta^*} G(Z,\tilde{Z}).$$

Similar to the proof for the first-order update, there exists some $\eta \in [0,1]$ and a parameter $\theta^*_\eta = \theta^* - \eta H^{-1}_{\theta^*} G(Z,\tilde{Z})$ such that

$$\begin{aligned}\nabla L\big(\theta^*{}_{Z\to\tilde{Z}}, D'\big) &= \nabla L\big(\theta^*, D'\big) \\ &\quad + \nabla^2 L\big(\theta^* + \eta(\theta^*{}_{Z\to\tilde{Z}} - \theta^*), D'\big)\big(\theta^*{}_{Z\to\tilde{Z}} - \theta^*\big) \\ &= \nabla L\big(\theta^*, D'\big) - H_{\theta^*_\eta} H^{-1}_{\theta^*} G(Z,\tilde{Z}).\end{aligned}$$

Using again that $\nabla L\big(\theta^*, D'\big) = G(Z,\tilde{Z})$ we arrive at

$$\begin{aligned}\big\|\nabla L\big(\theta^*{}_{Z\to\tilde{Z}}, D'\big)\big\|_2 &= \big\|G(Z,\tilde{Z}) - H_{\theta^*{}_\eta} H^{-1}_{\theta^*} G(Z,\tilde{Z})\big\|_2 \\ &= \big\|\big(H_{\theta^*} - H_{\theta^*{}_\eta}\big) H^{-1}_{\theta^*} G(Z,\tilde{Z})\big\|_2 \\ &\leq \big\|H_{\theta^*} - H_{\theta^*{}_\eta}\big\|_2 \big\|H^{-1}_{\theta^*}\big\|_2 \big\|G(Z,\tilde{Z})\big\|_2\end{aligned}$$

The λ-strong convexity of L ensures that $\|H^{-1}_{\theta^*}\|_2 \leq \frac{1}{\lambda}$. In addition to $\|G(Z,\tilde{Z})\|_2 \leq M\gamma_z|Z|$, it remains to bound the difference between the Hessians. Using the Lipschitz continuity of the gradient $\nabla^2\ell$ for $z \in D'$, we first get

$$\begin{aligned}\big\|\nabla^2\ell\big(z,\theta^*\big) - \nabla^2\ell\big(z,\theta^*{}_\eta\big)\big\|_2 &\leq \gamma''\big\|\theta^* - \theta^*_\eta\big\|_2 \\ &\leq \gamma''\big\|H^{-1}_{\theta^*}\big\|_2 \big\|G(Z,\tilde{Z})\big\|_2\end{aligned}$$

and then for the Hessians obtain

$$\begin{aligned}\big\|H_{\theta^*} - H_{\theta^*_\eta}\big\|_2 &= \sum_{z\in D} \big\|\nabla^2\ell\big(z,\theta^*\big) - \nabla^2\ell\big(z,\theta^*_\eta\big)\big\|_2 \\ &\leq n\gamma''\big\|H^{-1}_{\theta^*}\big\|_2 \big\|G(Z,\tilde{Z})\big\|_2.\end{aligned}$$

Combining all results finally yields the theoretical bound for the second-order update of our

approach

$$\begin{aligned}\left\|\nabla L\left(\theta^{*}{}_{Z\to\tilde{Z}}, D'\right)\right\|_2 &\leq \left\|H_{\theta^*} - H_{\theta^*_\eta}\right\|_2 \left\|H_{\theta^*}^{-1}\right\|_2 \left\|G(Z,\tilde{Z})\right\|_2 \\ &\leq n\gamma''\left\|H_{\theta^*}^{-1}\right\|_2^2 \left\|G(Z,\tilde{Z})\right\|_2^2 \\ &\leq \gamma''\left(\frac{M\gamma_z}{\lambda}\right)^2 n|Z|^2\end{aligned}$$

□

Theorem 3 Let $\mathcal{A}$ be the learning algorithm that returns the unique minimum of $L_{\mathbf{b}}(\theta; D')$ and let $\mathcal{U}$ be an unlearning method that produces a model $\theta_{\mathcal{U}}$. If $\|\nabla L(\theta_{\mathcal{U}}; D')\|_2 \leq \varepsilon'$ for some $\varepsilon' > 0$ we have the following guarantees.

1. If $\mathbf{b}$ is drawn from a distribution with density $p(\mathbf{b}) = e^{-\frac{\varepsilon}{\varepsilon'}\|\mathbf{b}\|_2}$ then $\mathcal{U}$ performs ε-certified unlearning for $\mathcal{A}$.

2. If $p \sim \mathcal{N}(0, c\varepsilon'/\varepsilon)^d$ for some $c > 0$ then $\mathcal{U}$ performs (ε, δ)-certified unlearning for $\mathcal{A}$ with $\delta = 1.5e^{-c^2/2}$.

Proof. The proofs work similarly to the sensitivity proofs for differential privacy as presented in Dwork and Roth [83].

1. Given $\mathbf{b}_1$ and $\mathbf{b}_2$ with $\|\mathbf{b}_1 - \mathbf{b}_2\|_2 \leq \varepsilon'$. By the construction of the density p, we have

$$\frac{p(\mathbf{b}_1)}{p(\mathbf{b}_2)} = e^{-\frac{\varepsilon}{\varepsilon'}\left(\|\mathbf{b}_1\|_2 - \|\mathbf{b}_2\|_2\right)} \leq e^{\frac{\varepsilon}{\varepsilon'}\left(\|\mathbf{b}_1 - \mathbf{b}_2\|_2\right)} \leq e^{\varepsilon}.$$

   If we now apply Theorem 2, the claim follows.

2. The second proof is similar to Theorem 3.22 in Dwork and Roth [83] using $\Delta_2(f) = \varepsilon'$ which yields that with probability at least $1 - \delta$ we have $e^{-\varepsilon} \leq \frac{p(\mathbf{b}_1)}{p(\mathbf{b}_2)} \leq e^{\varepsilon}$. Applying Theorem 2 afterwards again finalizes the proof.

□

Lemma 2 If the gradient residual of a single update step is bounded by C then the gradient residual after T consecutive update steps is bounded by TC.

Proof. The claim follows by mathematical induction: Recall that θ^* is an exact solution of $L(\theta, D)$. We define $\theta^{(0)} := \theta^*$ and the approximate solution after t steps as $\theta^{(t)} := \theta^{(t-1)} + \Delta^{(t)}$, where $\Delta^{(t)}$ is our first-order or second-order update. If we denote the gradient residual

after t steps by $\mathbf{u}^{(t)} = \|\nabla L\big(\theta^{(t)}, D^{t+1}\big)\|_2$ then Theorem 1 states that $\|u^{(1)}\| \leq C$ and thus the base case, where $D^{(t+1)}$ is the dataset where features or labels have been changed or revoked. For the induction step, consider the modified loss function

$$\tilde{L}\big(\theta, D^{(t+1)}\big) = L\big(\theta, D^{(t+1)}\big) - \theta^T \mathbf{u}^{(t)}.$$

By construction, $\theta^{(t)}$ optimizes $\tilde{L}$ since

$$\nabla \tilde{L}\big(\theta^{(t)}, D^{(t+1)}\big) = \nabla L\big(\theta^{(t)}, D^{(t+1)}\big) - \mathbf{u}^{(t)} = 0.$$

This allows to apply Theorem 1 to $\tilde{L}$ and yields the bound $\|\nabla \tilde{L}\big(\theta^{(t)} + \Delta^{(t)}, D^{(t+2)}\big)\|_2 \leq C$.

In other words, we have

$$\|\nabla \tilde{L}\big(\theta^{(t+1)}, D^{(t+2)}\big)\|_2 = \|\nabla L\big(\theta^{(t+1)}, D^{(t+2)}\big) - \mathbf{u}^{(t)}\|_2 = \|\mathbf{u}^{(t+1)} - \mathbf{u}^{(t)}\|_2 \leq C,$$

however by the induction statement we also know that $\|\mathbf{u}^{(t)}\|_2 \leq tC$ and therefore get

$$\|\mathbf{u}^{(t+1)}\|_2 = \|\mathbf{u}^{(t+1)} + \mathbf{u}^{(t)} - \mathbf{u}^{(t)}\|_2 \leq \|\mathbf{u}^{(t+1)} - \mathbf{u}^{(t)}\|_2 + \|\mathbf{u}^{(t)}\|_2 \leq (t+1)C.$$

□

References

[1] Regulation 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data. *Official Journal of the European Union*, 119:1–88, 2016.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 308–318, 2016.

[3] Julius Adebayo, Justin Gilmer, Ian Goodfellow, and Been Kim. Local explanation methods for deep neural networks lack sensitivity to parameter values. *arXiv:1810.03307*, 2018.

[4] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian J. Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 9505–9515, 2018.

[5] Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 700–712, 2020.

[6] Julius Adebayo, Michael Muelly, Harold Abelson, and Been Kim. Post hoc explanations may be ineffective for detecting unknown spurious correlation. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2022.

[7] Advanced Micro Devices, Inc. Vitis AI, 2023. Available at https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html.

[8] Advanced Micro Devices, Inc. Vitis AI DPU, 2023. Available at https://github.com/Xilinx/Vitis-AI/tree/master/dpu.

[9] Chirag Agarwal and Anh Gia-Tuan Nguyen. Explaining image classifiers by removing input features using generative models. In *Proc. of the Asian Conference on Computer Vision*, pages 101–118, 2020.

[10] Chirag Agarwal, Satyapriya Krishna, Eshika Saxena, Martin Pawelczyk, Nari Johnson, Isha Puri, Marinka Zitnik, and Himabindu Lakkaraju. Openxai: Towards a transparent evaluation of model explanations. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 15784–15799, 2022.

[11] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research (JMLR)*, page 4148–4187, 2017.

[12] Md. Mahbub Alam, Shahin Tajik, Fatemeh Ganji, Mark M. Tehranipoor, and Domenic Forte. RAM-Jam: Remote Temperature and Voltage Fault Attack on FPGAs using Memory Collisions. In *Proc. of the IEEE Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 48–55, 2019.

[13] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. iNNvestigate neural networks! *arxiv:1808.04260*, 2018.

[14] Nasser Aldaghri, Hessam Mahdavifar, and Ahmad Beirami. Coded machine unlearning. *IEEE Access*, 9:88137 – 88150, 2021.

[15] Victor Manuel Alvarez. Yara – the pattern matching swiss knife for malware researchers. https://virustotal.github.io/yara/. visited March 2021.

[16] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus Gross. Towards better understanding of gradient-based attribution methods for deep neural networks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[17] Christopher Anders, Plamen Pasliev, Ann-Kathrin Dombrowski, Klaus-Robert Müller, and Pan Kessel. Fairwashing explanations with off-manifold detergent. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 314–323, 2020.

[18] Christopher J. Anders, Leander Weber, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Finding and removing clever hans: Using explanation methods to debug and improve deep models. *Information Fusion*, 77: 261–295, 2019.

[19] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin: Efficient and explainable detection of Android malware in your pocket. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, February 2014.

[20] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*, August 2022.

[21] Daniel Christopher Arp. *Efficient and Explainable Detection of Mobile Malware with Machine Learning*. PhD thesis, Technische Universität Braunschweig, Sep 2019.

[22] Leila Arras, Franziska Horn, Grégoire Montavon, Klaus-Robert Müller, and Wojciech Samek. What is relevant in a text document?: An interpretable machine learning approach. *PLoS ONE*, 12, 2017.

[23] Peter Asaro. On banning autonomous weapon systems: human rights, automation, and the dehumanization of lethal decision-making. *International Review of the Red Cross*, 94(886):687–709, 2012.

[24] Joshua Attenberg, Kilian Weinberger, Anirban Dasgupta, Alex Smola, and Martin Zinkevich. Collaborative email-spam filtering with the hashing trick. In *Conference on Email and Anti-Spam (CEAS)*, 2009.

[25] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLoS ONE*, 10(7), July 2015.

[26] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B. Grosse. If influence functions are the answer, then what is the question? In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 17953–17967, 2022.

[27] David Balduzzi, Marcus Frean, Lennox Leary, J P Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proc. of the International Conference on Machine Learning (ICML)*, ICML'17, page 342–350, 2017.

[28] Shumeet Baluja and Ian Fischer. Learning to attack: Adversarial transformation networks. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, pages 2687–2695, 2018.

[29] Elnaz Barshan, M. Brunet, and G. Dziugaite. Relatif: Identifying explanatory training examples via relative influence. In *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1899–1909, 2020.

[30] Samyadeep Basu, Xuchen You, and Soheil Feizi. On second-order group influence functions for black-box predictions. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 715–724, 2020.

[31] Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2021.

[32] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, behavior-based malware clustering. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2009.

[33] Georg T. Becker, Francesco Regazzoni, Christof Paar, and Wayne P. Burleson. Stealthy dopant-level hardware trojans. In *Cryptographic Hardware and Embedded Systems International Workshop*, pages 197–214, 2013.

[34] Umang Bhatt, Adrian Weller, and José M. F. Moura. Evaluating and aggregating feature-based model explanations. In *Proc. of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, 2021. ISBN 9780999241165.

[35] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proc. of International Conference on Machine Learning (ICML)*, page 1467–1474, 2012.

[36] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[37] M. Bishop. *Computer security: Art and science*. Addison-Wesley, 2003.

[38] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 141–159, 2021.

[39] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. 2004.

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

[41] Marc-Etienne Brunet, Colleen Alkalay-Houlihan, A. Anderson, and R. Zemel. Understanding the origins of bias in word embeddings. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 803–811, 2019.

[42] Zoya Bylinskii, Tilke Judd, Aude Oliva, Antonio Torralba, and Frédo Durand. What do different evaluation metrics tell us about saliency models? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 740–757, 2016.

[43] Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 463–480, 2015.

[44] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, page 3–14, 2017.

[45] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 39–57, 2017.

[46] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Proc. of the USENIX Security Symposium*, pages 267–284, 2019.

[47] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, and Adam Roberts. Extracting training data from large language models. In *Proc. of the USENIX Security Symposium*, pages 2633–2650, 2021.

[48] Saikat Chakraborty, Rahul Krishna, Yangruibo Ding, and Baishakhi Ray. Deep learning based vulnerability detection: Are we there yet? *IEEE Transactions on Software Engineering*, 48:3280–3296, 2020.

[49] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. In *Proc. of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 839–847, 2018.

[50] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 289–296, 2008.

[51] Kamalika Chaudhuri, Claire Monteleoni, and Anand D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research (JMLR)*, page 1069–1109, 2011.

[52] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian M. Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. In *Proc. of the Workshop on Artificial Intelligence Safety*, 2019.

[53] Hongge Chen, Si Si, Yang Li, Ciprian Chelba, Sanjiv Kumar, Duane Boning, and Cho-Jui Hsieh. Multi-stage influence function. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 12732–12742, 2020.

[54] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proc. of the International Joint Conference on Artificial Intelligence*, pages 4658–4664, 2019.

[55] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, pages 15–26, 2017.

[56] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arxiv:1712.05526*, 2017.

[57] Animesh Chhotaray, Adib Nahiyan, Thomas Shrimpton, Domenic Forte, and Mark M. Tehranipoor. Standardizing bad cryptographic practice: A teardown of

the IEEE standard for protecting electronic-design intellectual property. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 1533–1546, 2017.

[58] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, October 2014.

[59] Theo Chow, Zeliang Kan, Lorenz Linhardt, Daniel Arp, Lorenzo Cavallaro, and Fabio Pierazzi. Drift forensics of malware classifiers. In *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, pages 197–207, 2023.

[60] Zheng Leong Chua, Shiqi Shen, Prateek Saxena, and Zhenkai Liang. Neural nets can learn function type signatures from binaries. In *Proc. of the USENIX Security Symposium*, pages 99–116, 2017.

[61] Joseph Clements and Yingjie Lao. Hardware Trojan Attacks on Neural Networks. *arxiv:1806.05768*, 2018.

[62] Tim Clifford, Ilia Shumailov, Yiren Zhao, Ross J. Anderson, and Robert D. Mullins. Impnet: Imperceptible and blackbox-undetectable backdoors in compiled neural networks. *arxiv:2210.00108*, 2022.

[63] Jeremy M. Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 1310–1320, 2019.

[64] R. Dennis Cook and Sanford Weisberg. Residuals and influence in regression. *New York: Chapman and Hall*, 1982.

[65] Ian C. Covert, Scott Lundberg, and Su-In Lee. Explaining by removing: A unified framework for model explanation. *Journal of Machine Learning Research*, 22, 2021.

[66] Ekin Cubuk, Barret Zoph, Samuel Schoenholz, and Quoc Le. Intriguing properties of adversarial examples. *arXiv:1711.02846*, 11 2017.

[67] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 6967–6976. 2017.

[68] Jessica Dai, Sohini Upadhyay, U. Aïvodji, Stephen H. Bach, and Himabindu Lakkaraju. Fairness via explanation quality: Evaluating disparities in the quality of post hoc explanations. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pages 203–214, 2022.

[69] A. Datta, S. Sen, and Y. Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 598–617, 2016.

[70] Arturo de la Escalera andchat Jose M. Armingol and Mario Mata. Traffic sign recognition and analysis for intelligent vehicles. *Image Vis. Comput.*, 21(3):247–258, 2003.

[71] D. Defays. An efficient algorithm for a complete link method. *The Computer Journal*, 20(4):364–366, 01 1977.

[72] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[73] Damien Desfontaines and Balázs Pejó. Sok: Differential privacies. *Proceedings on Privacy Enhancing Technologies*, pages 288–313, 2020.

[74] Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, pages 83:1–83:5, 2016.

[75] Artem Dinaburg, Paul Royal, Monirul I. Sharif, and Wenke Lee. Ether: malware analysis via hardware virtualization extensions. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 51–62, 2008.

[76] Bao Gia Doan, Ehsan Abbasnejad, and Damith C. Ranasinghe. Februus: Input purification defense against trojan attacks on deep neural network systems. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, page 897–912, 2020.

[77] Ann Kathrin Dombrowski, Maximilian Alber, Christopher J. Anders, Marcel Ackermann, Klaus-Robert Müller, and Pan Kessel. Explanations can be manipulated and geometry is to blame. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 13567–13578, 2019.

[78] Arthur Drichel and Ulrike Meyer. False sense of security: Leveraging xai to analyze the reasoning and true performance of context-less dga classifiers. In *Proc. of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, page 330–345, 2023.

[79] Arthur Drichel, Nils Faerber, and Ulrike Meyer. First step towards explainable dga multiclass classification. *Proc. of the International Conference on Availability, Reliability and Security*, pages 25:1–25:13, 2021.

[80] Dheeru Dua and Casey Graff. UCI machine learning repository. census income data set., 2017.

[81] Dheeru Dua and Casey Graff. UCI machine learning repository. diabetis data set., 2017.

[82] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming*, pages 1–12, 2006.

[83] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, page 211–407, 2014.

[84] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2):1–42, 2012.

[85] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. Verifiable and provably secure machine unlearning. *arXiv:2210.09126*, 2022.

[86] Nikolay Elenkov. *Android Security Internals: An In-Depth Guide to Android's Security Architecture*. No Starch Press, USA, 1st edition, 2014.

[87] Dumitru Erhan, Y. Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network. *Technical Report, Univeristé de Montréal*, 2009.

[88] Andre Esteva, Brett Kuprel, Roberto A. Novoa, Justin Ko, Susan M. Swetter, Helen M. Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[89] Ming Fan, Wenying Wei, Xiaofei Xie, Yang Liu, Xiaohong Guan, and Ting Liu. Can we trust your explanations? sanity checks for interpreters in android malware analysis. *IEEE Transactions on Information Forensics and Security*, 16:838–853, 2020.

[90] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 2962–2970, 2015.

[91] R. C. Fong and A. Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, pages 3449–3457. IEEE Computer Society, 2017.

[92] John Foremost. DroidDream mobile malware. https://www.virusbulletin.com/virusbulletin/2012/03/droiddream-mobile-malware, 2012. (Online; accessed 09-January-2024).

[93] Feisi Fu and Wenchao Li. Sound and complete neural network repair with minimality and locality guarantees. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2022.

[94] Tom Ganz, Martin Härterich, Alexander Warnecke, and Konrad Rieck. Explaining graph neural networks for vulnerability discovery. In *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, November 2021.

[95] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith Chinthana Ranasinghe, and Surya Nepal. STRIP: a defence against trojan attacks on deep neural networks. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 113–125, 2019.

[96] Hugo Gascon, Bernd Grobauer, Thomas Schreck, Lukas Rist, Daniel Arp, and Konrad Rieck. Mining attributed graphs for threat intelligence. In *Proc. of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 15–22, March 2017.

[97] Amirata Ghorbani, Abubakar Abid, and James Y. Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2019.

[98] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael A. Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. *IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, 2018.

[99] A. Ginart, Melody Y. Guan, G. Valiant, and J. Zou. Making AI forget you: Data deletion in machine learning. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3513–3526, 2019.

[100] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[101] Aditya Golatkar, Alessandro Achille, Avinash Ravichandran, Marzia Polito, and Stefano Soatto. Mixed-privacy forgetting in deep networks. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[102] T.F. Gonzàlez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science 38*, pages 293–306, 1985.

[103] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[104] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[105] Alex Graves. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

[106] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick D. McDaniel. Adversarial examples for malware detection. In *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, pages 62–79, 2017.

[107] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[108] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Comput. Surv.*, 51(5), aug 2018.

[109] Chuan Guo, Jacob R. Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q. Weinberger. Simple black-box adversarial attacks. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 97, pages 2484–2493, 2019.

[110] Chuan Guo, Tom Goldstein, Awni Y. Hannun, and Laurens van der Maaten. Certified data removal from machine learning models. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 3822–3831, 2020.

[111] Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arxiv:2012.15781*, 2020.

[112] Wenbo Guo, Dongliang Mu, Jun Xu, Purui Su, Gang Wang, and Xinyu Xing. LEMNA: Explaining deep learning based security applications. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 364–379, 2018.

[113] F. Hampel. The influence curve and its role in robust estimation. In *Journal of the American Statistical Association*, 1974.

[114] Dongqi Han, Zhiliang Wang, Wenqi Chen, Ying Zhong, Su Wang, Han Zhang, Jiahai Yang, Xingang Shi, and Xia Yin. Deepaid: Interpreting and improving deep learning-based anomaly detection in security applications. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, 2021.

[115] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[116] Tessa Han, Suraj Srinivas, and Himabindu Lakkaraju. Which explanation should i choose? a function approximation perspective to characterizing post hoc explanations. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 5256–5268, 2022.

[117] Stefan Harmeling, Guido Dornhege, David Tax, Frank Meinecke, and Klaus-Robert Müller. From outliers to prototypes: Ordering data. *Neurocomput.*, 69(13–15):1608–1618, 2006.

[118] Babak Hassibi, David G. Stork, and Gregory J. Wolff. Optimal brain surgeon: Extensions and performance comparison. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 263–270, 1993.

[119] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2015.

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

[121] Sanghyun Hong, Pietro Frigo, Yigitcan Kaya, Cristiano Giuffrida, and Tudor Dumitras. Terminal Brain Damage: Exposing the Graceless Degradation in Deep Neural Networks Under Hardware Fault Attacks. In *Proc. of the USENIX Security Symposium*, pages 497–514, 2019.

[122] Sara Hooker, Dumitru Erhan, Pieter-Jan Kindermans, and Been Kim. A benchmark for interpretability methods in deep neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 9734–9745, 2019.

[123] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *Proc. of the IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013.

[124] Cheng-Yu Hsieh, Chih-Kuan Yeh, Xuanqing Liu, Pradeep Ravikumar, Seungyeon Kim, Sanjiv Kumar, and Cho-Jui Hsieh. Evaluations and methods for explanation through robustness analysis. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2019.

[125] Xin Hu, Sandeep Bhatkar, Kent Griffin, and Kang G. Shin. Mutantx-s: Scalable malware clustering based on static feature. In *Proc. of the USENIX Annual Technical Conference*, pages 187–198, 2013.

[126] Xing Hu, Yang Zhao, Lei Deng, Ling Liang, Pengfei Zuo, Jing Ye, Yingyan Lin, and Yuan Xie. Practical Attacks on Deep Neural Networks by Memory Trojaning. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(6):1230–1243, 2021.

[127] Wenyi Huang and Jack W. Stokes. MtNet: A multi-task neural network for dynamic malware classification. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 399–418, 2016.

[128] Xijie Huang, Moustafa Farid Alzantot, and Mani B. Srivastava. Neuroninspect: Detecting backdoors in neural networks via output explanations. *arxiv:1911.07399*, 2019.

[129] Lawrence J. Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2:193–218, 1985.

[130] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. Euphony: Harmonious

unification of cacophonous anti-virus vendor labels for Android malware. In *Proc. of the International Conference on Mining Software Repositories (MSR)*, pages 425–435, 2017.

[131] Médéric Hurier, Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 142–162, 2016.

[132] Maximilian Idahl, Lijun Lyu, Ujwal Gadiraju, and Avishek Anand. Towards benchmarking the utility of explanations for model debugging. *arxiv:2105.04505*, 2021.

[133] IEEE Design Automation Standards Committee (DASC). *IEEE 1735-2014 - Recommended Practice for Encryption and Management of Electronic Design Intellectual Property (IP)*. 2015.

[134] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 2142–2151, 2018.

[135] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2019.

[136] Zachary Izzo, Mary Anne Smart, Kamalika Chaudhuri, and James Zou. Approximate data deletion from machine learning models. In *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 2008–2016, 2021.

[137] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, 2018.

[138] Jiyong Jang, David Brumley, and Shobha Venkataraman. BitShred: feature hashing malware for scalable triage and semantic analysis. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 309–320, 2011.

[139] Xuxian Jiang. Security Alert: New sophisticated Android malware DroidKungFu found in alternative chinese App markets. https://www.csc2.ncsu.edu/faculty/xjiang4/DroidKungFu.html, 2011. (Online; accessed 14-February-2019).

[140] Xuxian Jiang. Security Alert: New Android malware GoldDream found in alternative app markets. https://www.csc2.ncsu.edu/faculty/xjiang4/GoldDream/, 2011. (Online; accessed 14-February-2019).

[141] Nick Kanopoulos, Nagesh Vasanthavada, and Robert L Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of solid-state circuits*, 23 (2):358–367, 1988.

[142] Alexandros Kapravelos, Yan Shoshitaishvili, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Revolver: An automated approach to the detection of evasive web-based malware. In *Proc. of the USENIX Security Symposium*, pages 637–651, August 2013.

[143] J. Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *Annals of Mathematical Statistics*, 23:462–466, 1952.

[144] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proc. of the ACM International Conference on Management of Data*, page 193–204, 2011.

[145] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie J. Cai, James Wexler, Fernanda B. Viégas, and Rory Sayres. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In *Proc. of the International Conference on Machine Learning (ICML)*, pages 2673–2682, 2018.

[146] Pieter-Jan Kindermans, Sara Hooker, Julius Adebayo, Maximilian Alber, Kristof T. Schütt, Sven Dähne, D. Erhan, and Been Kim. The (un)reliability of saliency methods. In *Explainable AI*, 2017.

[147] Pieter-Jan Kindermans, Kristof T. Schütt, Maximilian Alber, Klaus-Robet Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[148] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[149] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Proccessing Systems (NIPS)*, volume 28, pages 2575–2583, 2015.

[150] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 1885–1894, 2017.

[151] Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 5255–5265, 2019.

[152] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao yong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proc. of the USENIX Security Symposium*, pages 351–366, 2009.

[153] Stefan Kolek, Duc Anh Nguyen, Ron Levie, Joan Bruna, and Gitta Kutyniok. *A Rate-Distortion Framework for Explaining Black-Box Model Decisions*, pages 91–115. Springer International Publishing, 2022.

[154] Vivek Kothari, Edgar Liberis, and Nicholas D. Lane. The final frontier: Deep learning in space. In *Proc. of the International Workshop on Mobile Computing Systems and Applications*, pages 45–49, 2020.

[155] Satyapriya Krishna, Tessa Han, Alex Gu, Javin Pombra, Shahin Jabbari, Steven Wu, and Himabindu Lakkaraju. The disagreement problem in explainable machine learning: A practitioner's perspective. *arxiv:2202.01602*, 2022.

[156] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 1106–1114. Curran Associates, Inc., 2012.

[157] S N Kumar, A. Fred, Ajay Kumar Haridhas, and S. Varghese. Medical image edge detection using gauss gradient operator. *Journal of Pharmaceutical Sciences and Research*, 9:695–704, 2017.

[158] Sebastian Lapuschkin, Stephan Wäldchen, Alexander Binder, Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. Unmasking clever hans predictors and assessing what machines really learn. *Nature Communications*, 10, 2019.

[159] Y.A. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.J. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[160] Yann LeCun and Yoshua Bengio. Word-level training of a handwritten word recognizer based on convolutional neural networks. In *Proc. of the International Conference on Pattern Recognition (ICPR)*, pages 88–92, 1994.

[161] Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 598–605, 1989.

[162] Mathias Lécuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. Certified robustness to adversarial examples with differential privacy. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 656–672, 2019.

[163] He Li, Qiang Liu, and Jiliang Zhang. A survey of hardware Trojan threat and defense. *Integr.*, 55:426–437, 2016.

[164] Wenshuo Li, Jincheng Yu, Xuefei Ning, Pengjun Wang, Qi Wei, Yu Wang, and Huazhong Yang. Hu-Fu: Hardware and Software Collaborative Attack Framework Against Neural Networks. In *Proc. of the IEEE Computer Society Annual Symposium on VLSI*, pages 482–487, 2018.

[165] Yuanchun Li, Jiayi Hua, Haoyu Wang, Chunyang Chen, and Yunxin Liu. Deeppayload: Black-box backdoor attack on deep learning models through neural payload injection. In *Proc. of the IEEE/ACM International Conference on Software Engineering*, pages 263–274, 2021.

[166] Zhen Li, Deqing Zou, Shouhuai Xu, Xinyu Ou, Hai Jin, Sujuan Wang, Zhijun Deng, and Yuyi Zhong. Vuldeepecker: A deep learning-based system for vulnerability detection. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[167] Cong Liao, Haoti Zhong, Anna Cinzia Squicciarini, Sencun Zhu, and David J. Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *Proc. of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2018.

[168] Martina Lindorfer, Clemens Kolbitsch, and Paolo Milani Comparetti. Detecting environment-sensitive malware. In *Proc. of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 338–357, 2011.

[169] Wenye Liu, Chip-Hong Chang, Fan Zhang, and Xiaoxuan Lou. Imperceptible Misclassification Attack on Deep Learning Accelerator by Glitch Injection. In *Proc. of the ACM/IEEE Design Automation Conference*, pages 1–6, 2020.

[170] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, 2018.

[171] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[172] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3288–3298, 2017.

[173] Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[174] Yang Lu, Wenbo Guo, Xinyu Xing, and William Stafford Noble. Robust decoy-enhanced saliency maps. *arXiv:2002.00526*, 2020.

[175] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 4765–4774, 2017.

[176] Hua Ma, Huming Qiu, Yansong Gao, Zhi Zhang, Alsharif Abuadbba, Anmin Fu, Said F. Al-Sarawi, and Derek Abbott. Quantization backdoors to deep learning models. *arxiv:2108.09187*, 2021.

[177] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[178] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[179] McAfee. Android/FakeInstaller.L. https://home.mcafee.com/virusinfo/, 2012. (Online; accessed 1-August-2018).

[180] Niall McLaughlin, Jesús Martínez del Rincón, BooJoong Kang, Suleiman Y. Yerima, Paul C. Miller, Sakir Sezer, Yeganeh Safaei, Erik Trickel, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Deep android malware detection. In *Proc. of the ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 301–308, 2017.

[181] Shike Mei and Xiaojin Zhu. Using machine teaching to identify optimal training-set attacks on machine learners. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, 2015.

[182] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *arxiv:1803.08240*, 2018.

[183] V. Metsis, G. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes - which naive bayes? In *Proc. of Conference on Email and Anti-Spam (CEAS)*, 2006.

[184] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proc. of the International Conference on Learning Representations (ICLR Workshop)*, 2013.

[185] Smitha Milli, Ludwig Schmidt, Anca D. Dragan, and Moritz Hardt. Model reconstruction from model explanations. In *Proc. of the Conference on Fairness, Accountability, and Transparency (FAT)*, pages 1–9, 2019.

[186] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2022.

[187] Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Variational dropout sparsifies deep neural networks. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 2498–2507, 2017.

[188] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. *Layer-Wise Relevance Propagation: An Overview*. Springer International Publishing, 2019.

[189] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017.

[190] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94, 2016.

[191] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2574–2582, June 2016.

[192] Niels J. S. Mørch, Ulrik Kjems, Lars Kai Hansen, Claus Svarer, Ian Law, Benny Lautrup, Stephen C. Strother, and Kelly Rehm. Visualization of neural networks using saliency maps. In *Proc. of International Conference on Neural Networks (ICNN)*, pages 2085–2090, 1995.

[193] Andreas Moser, Christopher Kruegel, and Engin Kirda. Limits of static analysis for malware detection. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 421–430, 2007.

[194] Rijoy Mukherjee and Rajat Subhra Chakraborty. Novel Hardware Trojan Attack on Activation Parameters of FPGA-Based DNN Accelerators. *IEEE Embed. Syst. Lett.*, 14(3):131–134, 2022.

[195] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C. Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. *Proc. of ACM Workshop on Artificial Intelligence and Security (AISEC)*, pages 27–38, 2017.

[196] Krishna Kanth Nakka and Mathieu Salzmann. Learning transferable adversarial perturbations. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 13950–13962, 2021.

[197] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-delete: Gradient-based methods for machine unlearning. In *Proc. of the International Conference on Algorithmic Learning Theory (ALT)*, pages 931–962, 2021.

[198] Matthias Neugschwandtner, Paolo Milani Comparetti, Grégoire Jacob, and Christopher Kruegel. Forecast: skimming off the malware cream. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 11–20, 2011.

[199] Tuan Anh Nguyen and Anh Tuan Tran. Input-aware dynamic backdoor attack. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3454–3464, 2020.

[200] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, 2e edition, 2006.

[201] Maximilian Noppel, Lukas Peter, and Christian Wressnegger. Disguising attacks with explanation-aware backdoors. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 664–681, 2023.

[202] Tolulope A. Odetola, Hawzhin Raoof Mohammed, and Syed Rafay Hasan. A Stealthy Hardware Trojan Exploiting the Architectural Vulnerability of Deep Learning Architectures: Input Interception Attack (IIA). *arxiv:1911.00783*, 2019.

[203] Nicolas Papernot, Patrick Mcdaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. *Proc. of the ACM Asia Conference on Computer and Communications Security (ASIA CCS)*, pages 506–519, 2016.

[204] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. SoK: Security and privacy in machine learning. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414, 2018.

[205] Nicolas Papernot, Patrick D. McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 399–414, 2018.

[206] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *Proc. of the British Machine Vision Conference (BMVC)*, pages 41.1–41.12, 2015.

[207] Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6(1), 1994.

[208] Roberto Perdisci and ManChon U. Vamo: towards a fully automated malware clustering validity analysis. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 329–338, 2012.

[209] Roberto Perdisci, Andrea Lanzi, and Wenke Lee. Classification of paked executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941–1946, 2008.

[210] Lukas Pirch, Alexander Warnecke, Christian Wressnegger, and Konrad Rieck. Tagvet: Vetting malware tags using explainable machine learning. In *Proc. of the European Workshop on System Security (EUROSEC)*, 2021.

[211] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[212] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge J. Belongie. Generative adversarial perturbations. *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4422–4431, 2017.

[213] David M. W. Powers. Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Research (JMLR)*, 2:37–63, 2011.

[214] Endres Puschner, Thorben Moos, Steffen Becker, Christian Kison, Amir Moradi, and Christof Paar. Red team vs. blue team: A real-world hardware trojan detection case study across four modern CMOS technology generations. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 56–74, 2023.

[215] Junyang Qiu, Jun Zhang, Wei Luo, Lei Pan, Surya Nepal, Yu Wang, and Yang Xiang. A3cm: Automatic capability annotation for Android malware. *IEEE Access*, 7: 147156–147168, 2019.

[216] Dima Rabadi and Sin G. Teo. Advanced windows methods on malware detection and classification. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 54–68, 2020.

[217] Kamiar Rahnama Rad and Arian Maleki. A scalable estimate of the extra-sample prediction error via approximate leave-one-out. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82, 2018.

[218] Aditi Raghunathan, Jacob Steinhardt, and Percy Liang. Certified defenses against adversarial examples. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[219] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? In *Proc. of the International Conference on Machine Learning (ICML)*, pages 5389–5400, 2019.

[220] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proc. of ACM SIGKDD International Conference On Knowledge Discovery and Data Mining (KDD)*, pages 1135–1144, 2016.

[221] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 32, pages 1527–1535, 2018.

[222] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. Technical Report 2009-18, Technische Universität Berlin, December 2009.

[223] Konrad Rieck, Philipp Trinius, Carsten Willems, and Thorsten Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security (JCS)*, 19(4):639–668, June 2011.

[224] Laura Rieger and Lars Kai Hansen. A simple defense against adversarial attacks on heatmap explanations. *arxiv:2007.06381*, 2020.

[225] Laura Rieger and Lars Kai Hansen. Irof: a low resource evaluation metric for explanation methods. *arxiv:2003.08747*, 2020.

[226] Herbert E. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[227] J-Michael Roberts. Virusshare.com. https://www.virusshare.com. visited January 2023.

[228] R. Rojas. *Neural Networks: A Systematic Approach*. Springer-Verlag, Berlin, Deutschland, 1996.

[229] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Annals of Mathematical Statistics*, 27:832–837, 1956.

[230] Daniel Rosenkrantz, Richard Stearns, and Philip II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6:563–581, 09 1977.

[231] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[232] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard E. Harang. Aloha: Auxiliary loss optimization for hypothesis augmentation. In *Proc. of the USENIX Security Symposium*, pages 303–320, 2019.

[233] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 11957–11965, 2020.

[234] Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28:2660–2673, 2015.

[235] Wojciech Samek, Gregoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Muller. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. 2019.

[236] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *arxiv:1804.11285*, 2018.

[237] T. Schnake, O. Eberle, J. Lederer, S. Nakajima, K. T. Schutt, K. Muller, and G. Montavon. Higher-order explanations of graph neural networks via relevant walks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7581–7596, nov 2022.

[238] Peter Schulam and Suchi Saria. Can you trust this prediction? auditing pointwise reliability after learning. In *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1022–1031, 2019.

[239] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *Proc. of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, pages 230–253, 2016.

[240] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 42–53, 2020.

[241] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from networks via gradient-based localization. In *Proc. of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, 2017.

[242] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 6106–6116, 2018.

[243] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John P. Dickerson, Christoph Studer, Larry S. Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3353–3364, 2019.

[244] L.S. Shapley. A value for n-person games. 1953.

[245] Mahmood Sharif, Lujo Bauer, and Michael K. Reiter. On the suitability of lp-norms for creating and preventing adversarial examples. *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1686–16868, 2018.

[246] Monirul I. Sharif, Vinod Yegneswaran, Hassen Saïdi, Phillip A. Porras, and Wenke Lee. Eureka: A framework for enabling static malware analysis. In *Proc. of the European Symposium on Research in Computer Security (ESORICS)*, pages 481–500, 2008.

[247] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *Proc. of the USENIX Security Symposium*, pages 611–626, 2015.

[248] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv:1605.01713*, 2016.

[249] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 3145–3153, 2017.

[250] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2014.

[251] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2015.

[252] Sahil Singla, Eric Wallace, Shi Feng, and Soheil Feizi. Understanding impacts of high-order loss approximations and features in deep learning interpretation. In *Proc. of the International Conference on Machine Learning (ICML)*, volume 97, pages 5848–5856, 2019.

[253] Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2020.

[254] Sergei Skorobogatov and Christopher Woods. Breakthrough silicon scanning discovers backdoor in military chip. In *Cryptographic Hardware and Embedded Systems International Workshop*, volume 7428, pages 23–40, 2012.

[255] Dylan Slack, Sophie Hilgard, Emily Jia, Sameer Singh, and Himabindu Lakkaraju. Fooling lime and shap: Adversarial attacks on post hoc explanation methods. In *Proc. of the AAAI/ACM Conference on Artificial Intelligence , Ethics, and Society (AIES)*, pages 180–186, 2019.

[256] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda B. Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv:1706.03825*, 2017.

[257] Alexander Smola and S.V.N. Vishwanathan. *Introduction to machine learning*. Cambridge University Press, 2008.

[258] Charles Smutz and Angelos Stavrou. Malicious PDF detection using metadata and structural features. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*, pages 239–248, 2012.

[259] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 305–316, 2010.

[260] Julian Speith, Florian Schweins, Maik Ender, Marc Fyrbiak, Alexander May, and Christof Paar. How not to protect your IP - an industry-wide break of IEEE 1735 implementations. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 1656–1671, 2022.

[261] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

[262] Suraj Srinivas, Akshayvarun Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *Proc. of the International Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 455–462, 2017.

[263] Cynthia Sturton, Matthew Hicks, David A. Wagner, and Samuel T. King. Defeating UCI: building stealthy and malicious hardware. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 64–77, 2011.

[264] Rui Sun, Tao Lei, Qi Chen, Zexuan Wang, Xiaogang Du, Weiqiang Zhao, and Asoke K. Nandi. Survey of image edge detection. *Frontiers in Signal Processing*, 2, 2022.

[265] Zhichuang Sun, Ruimin Sun, Long Lu, and Alan Mislove. Mind your weight(s): A large-scale study on insufficient machine learning model protection in mobile apps. In *Proc. of the USENIX Security Symposium*, pages 1955–1972, 2021.

[266] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proc. of International Conference on Machine Learning (ICML)*, pages 3319–3328, 2017.

[267] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 1017–1024, 2011.

[268] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 3104–3112, 2014.

[269] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2014.

[270] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[271] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[272] Di Tang, XiaoFeng Wang, Haixu Tang, and Kehuan Zhang. Demon in the variant: Statistical analysis of dnns for robust backdoor contamination detection. In *Proc. of the USENIX Security Symposium*, pages 1541–1558, 2021.

[273] Ryutaro Tanno, Melanie Fernandes Pradier, Aditya Nori, and Yingzhen Li. Repairing neural networks by leaving the right past behind. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 13132–13145, 2022.

[274] Ayush K Tarun, Vikram S Chundawat, Murari Mandal, and Mohan S. Kankanhalli. Fast yet effective machine unlearning. *IEEE transactions on neural networks and learning systems*, PP, 2021.

[275] Joe Tebelskis. *Speech Recognition using Neural Networks*. PhD thesis, Carnegie Mellon University, 1995.

[276] Mohammad Tehranipoor and Farinaz Koushanfar. A Survey of Hardware Trojan Taxonomy and Detection. *IEEE Des. Test Comput.*, 27(1):10–25, 2010.

[277] A. Thudi, G. Deza, V. Chandrasekaran, and N. Papernot. Unrolling sgd: Understanding factors influencing machine unlearning. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 303–319, 2022.

[278] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the necessity of auditable algorithmic definitions for machine unlearning. In *Proc. of the USENIX Security Symposium*, pages 4007–4022, August 2022.

[279] M. Caner Tol, Saad Islam, Berk Sunar, and Ziming Zhang. An optimization perspective on realizing backdoor injection attacks on deep neural networks in hardware. *arxiv:2110.07683*, 2021.

[280] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *Proc. of the USENIX Security Symposium*, pages 601–618, 2016.

[281] Florian Tramer, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2018.

[282] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 8011–8021, 2018.

[283] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arxiv:1912.02771*, 2019.

[284] Xabier Ugarte-Pedrero, Mariano Graziano, and Davide Balzarotti. A close look at a daily dataset of malware samples. *ACM Transactions on Privacy and Security*, 22(1), 2019.

[285] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2017.

[286] VirusTotal. Vt intelligence: Combine Google and Facebook and apply it to the field of malware. https://www.virustotal.com/gui/intelligence-overview, visited January 2023.

[287] VMRay GmbH. Malware analysis sandbox & malware detection software. https://www.vmray.com/products/analyzer-malware-sandbox/. visited January 2023.

[288] Nedim Šrndić and Pavel Laskov. Practical evasion of a learning-based classifier: A case study. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 197–211, 2014.

[289] Erik Štrumbelj and Igor Kononenko. Explaining prediction models and individual predictions with feature contributions. *Knowledge and Information Systems*, 41(3): 647–665, dec 2014.

[290] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 707–723, 2019.

[291] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: hardware-aware automated quantization with mixed precision. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8612–8620, 2019.

[292] Zifan Wang, Piotr (Peter) Mardziel, Anupam Datta, and Matt Fredrikson. Interpreting interpretations: Organizing attribution methods by criteria. In *Proc. of the International Conference on Computer Vision and Pattern Recognition Workshops (CVPR W)*, pages 48–55, 2020.

[293] Alexander Warnecke. Layerwise Relevance Propagation for LSTMs. https://github.com/alewarne/Layerwise-Relevance-Propagation-for-LSTMs, 2020.

[294] Alexander Warnecke. explain-mlsec. https://github.com/alewarne/explain-mlsec, 2020.

[295] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. Evaluating explanation methods for deep learning in security. In *Proc. of the IEEE European Symposium on Security and Privacy (EuroS&P)*, September 2020.

[296] Alexander Warnecke, Lukas Pirch, Christian Wressnegger, and Konrad Rieck. Machine unlearning of features and labels. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, February 2023.

[297] Alexander Warnecke, Julian Speith, Jan-Niklas Möller, Konrad Rieck, and Christof Paar. Evil from within: Machine learning backdoors through hardware trojans. *arXiv:2304.08411*, 2023.

[298] George D. Webster, Bojan Kolosnjaji, Christian von Pentz, Julian Kirsch, Zachary D. Hanif, Apostolis Zarras, and Claudia Eckert. Finding the needle: A study of the pe32 rich header and respective malware triage. In *Proc. of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 119–138, 2017.

[299] Feng Wei, Hongda Li, Ziming Zhao, and Hongxin Hu. xnids: Explaining deep learning-based network intrusion detection systems for active intrusion responses. In *Proc. of the USENIX Security Symposium*, pages 4337–4354, 2023.

[300] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Proccessing Systems (NIPS)*, pages 2074–2082, 2016.

[301] Georg Wicherski. peHash: A novel approach to fast malware clustering. In *Proc. of the USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[302] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. *arxiv:2001.03994*, 2020.

[303] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arxiv:2004.09602*, 2020.

[304] Yinjun Wu, Edgar Dobriban, and Susan B. Davidson. Deltagrad: Rapid retraining of machine learning models. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 10355–10366, 2020.

[305] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *Proc. of the International Joint Conference on Artificial Intelligence*, page 3905–3911, 2018.

[306] Han Xiao, Huang Xiao, and Claudia Eckert. Adversarial label flips attack on support vector machines. In *Proc. of European Conference on Artificial Intelligence (ECAI)*, pages 870–875, 2012.

[307] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 363–376, 2017.

[308] Xiaojun Xu, Qi Wang, Huichen Li, Nikita Borisov, Carl A. Gunter, and Bo Li. Detecting AI trojans using meta neural analysis. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 103–120, 2021.

[309] Mingfu Xue, Chongyan Gu, Weiqiang Liu, Shichao Yu, and Máire O'Neill. Ten years of hardware Trojans: a survey from the attacker's perspective. *IET Comput. Digit. Tech.*, 14(6):231–246, 2020.

[310] Fabian Yamaguchi, Nico Golde, Daniel Arp, and Konrad Rieck. Modeling and discovering vulnerabilities with code property graphs. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2014.

[311] Fabian Yamaguchi, Alwin Maier, Hugo Gascon, and Konrad Rieck. Automatic inference of search patterns for taint-style vulnerabilities. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[312] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Aliakbar Ahmadzadeh, Xinyu Xing, and Gang Wang. Cade: Detecting and explaining concept drift samples for security applications. In *Proc. of the USENIX Security Symposium*, pages 2327–2344, 2021.

[313] Mengjiao Yang and Been Kim. Benchmarking attribution methods with relative feature importance. *arXiv:1907.09701*, 2019.

[314] Jing Ye, Yu Hu, and Xiaowei Li. Hardware Trojan in FPGA CNN Accelerator. In *Proc. of the IEEE Asian Test Symposium*, pages 68–73, 2018.

[315] Chih-Kuan Yeh, Cheng-Yu Hsieh, Arun Sai Suggala, David I. Inouye, and Pradeep Ravikumar. On the (in)fidelity and sensitivity of explanations. In *Advances in Neural Information Proccessing Systems (NIPS)*, 2019.

[316] Santiago Zanella Béguelin, Lukas Wutschitz, Shruti Tople, Victor Rühle, Andrew Paverd, Olga Ohrimenko, Boris Köpf, and Marc Brockschmidt. Analyzing Information Leakage of Updates to Natural Language Models. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, pages 363–375, 2020.

[317] Matthew D. Zeiler. Adadelta: An adaptive learning rate method. *arxiv:1212.5701*, 2012.

[318] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. of European Conference on Computer Vision (ECCV)*, pages 818–833, 2014.

[319] Yi Zeng, Minzhou Pan, Hoang Anh Just, Lingjuan Lyu, Meikang Qiu, and Ruoxi Jia. Narcissus: A practical clean-label backdoor attack with limited information. In *Proc. of the ACM Conference on Computer and Communications Security (CCS)*, page 771–785, 2023.

[320] Jianming Zhang, Zhe L. Lin, Jonathan Brandt, Xiaohui Shen, and Stan Sclaroff. Top-down neural attention by excitation backprop. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1084–1102, 2016.

[321] Xinyang Zhang, Ningfei Wang, Hua Shen, Shouling Ji, Xiapu Luo, and Ting Wang. Interpretable deep learning under fire. In *Proc. of the USENIX Security Symposium*, pages 1659–1676, 2019.

[322] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proc. of the International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, 2016.

[323] Jianlong Zhou, Amir Hossein Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the quality of machine learning explanations: A survey on methods and metrics. *Electronics*, 2021.

[324] Yajin Zhou and Xuxian Jiang. Dissecting android malware: Characterization and evolution. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*, pages 95–109, 2012.

[325] Yaqin Zhou, Shangqing Liu, J. Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. *arxiv:1909.03496*, 2019.

[326] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Proc. of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 4596–4604, 2018.

[327] Luisa M. Zintgraf, Taco S. Cohen, Tameem Adel, and Max Welling. Visualizing deep neural network decisions: Prediction difference analysis. In *Proc. of the International Conference on Learning Representations*, 2017.