Saarland University
Faculty of Mathematics and Computer Science
Department of Computer Science

Why is Machine Learning Security so hard?

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

von
Kathrin Grosse

Saarbrücken, 2020

Tag des Kolloquiums:	20. Mai 2021
Dekan:	Prof. Dr. Thomas Schuster

Prüfungsausschuss:

Vorsitzender:	Prof. Dr. Dietrich Klakow
Berichterstattende:	Prof. Dr. Dr. H. C. Michael Backes
	Prof. Dr. Mario Fritz
Akademischer Mitarbeiter:	Dr. Robert Künnemann

Zusammenfassung

Die Zunahme von Rechenleistung und verfügbaren Datenmengen hat zu einer weitreichenden Anwendung von maschinellem Lernen geführt. Gleichzeitig birgt die breite Anwendung einer Technik immer Sicherheitsrisiken, da die Angriffsfläche entsprechend groß ist. Im Fall von maschinellem Lernen wurde gezeigt, dass eine kleine Veränderung der zu verarbeitenden Daten ausreicht, um das Resultat der Berechnungen zu verändern. Weiterhin kann ein Angreifer geistige Eigentumsrechte des Modellbesitzers verletzen, indem er Datenpunkte und Ausgabe des Algorithmus zusammenträgt und damit die Trainingsdaten inferiert oder ein neues Modell erzeugt. Auch das Verändern der Trainingsdaten ist wirkungsvoll, um dem Modell generell falsche Zusammenhänge oder im speziellen Hintertüren beizubringen. Weitere Angriffe sind bekannt, und nach aktuellem Kenntnisstand sind für keinen der Angriffe zufriedenstellende Lösungen bekannt. In dieser Arbeit wollen wir aus verschiedenen Perspektiven beleuchten, warum Sicherheit für maschinelles Lernen so komplex ist.

Wir beginnen mit sogenannten Adversarial Examples: Eingaben, die die Ausgabe eines trainierten Modells verändern. Diese zeigen uns, dass die Sicherheit von maschinellem Lernen ein inherentes Problem ist, nicht ein Fehler, der nur behoben werden muss. Ein weiteres Problem ist, dass Angriffe auf einem Modell oft direkt auf einem weiteren, unabhängigen Modell funktionieren. Wir wenden uns dann der Erkennung von Hintertüren in Modellen zu. Auch in Modellen, die ohne Hintertüren trainiert sind, manifestiert sich Verhalten, das auf Hintertüren hinweist. Die Sicherheitsfrage wird dann zu einer Frage nach böser Absicht, die schwierig beantwortet werden kann. Die Komplexität bei maschinellem Lernen ist allerdings noch weitreichender: Bibliotheken, die für maschinelles Lernen verwendet werden, sind oft groß und unübersichtlich. Wir belegen, wie eine Änderung in dem Initialisierungscode von neuronalen Netzen die trainierten Modelle erheblich verschlechtern kann. Der von uns beschriebene Angriff ist allerdings, wenn der Nutzer informiert ist, gut abwehrbar.

Bei vielen Angriffen reicht das Wissen über den Angriff als Verteidigung jedoch nicht aus. Zusätzliche Komplexität kommt ins Spiel, wenn nicht nur ein, sondern mehrere Angriffe berücksichtigt werden. Wir zeigen eine Konfiguration des Modells, mit der ein Angriff weniger effektiv ist—dann ist jedoch ein anderer Angriff um so effektiver. Unser letzter Punkt schließlich betrifft das Wissen über und Verständnis von maschinellem Lernen. Insbesondere zeigen wir hier die Wechselwirkung von der Entwicklung und dem Verstehen guter Algorithmen und Sicherheitsaspekten wie Verteidigungen, versuchten Angriffen und Angriffen.

Abstract

The increase of available data and computing power has fueled a wide application of machine learning (ML). At the same time, security concerns are raised: ML models were shown to be easily fooled by slight perturbations on their inputs. Furthermore, by querying a model and analyzing output and input pairs, an attacker can infer the training data or replicate the model, thereby harming the owner's intellectual property. Also, altering the training data can lure the model into producing specific or generally wrong outputs at test time. So far, none of the attacks studied in the field has been satisfactorily defended. In this work, we shed light on these difficulties.

We first consider classifier evasion or adversarial examples. The computation of such examples is an inherent problem, as opposed to a bug that can be fixed. We also show that adversarial examples often transfer from one model to another, different model. Afterwards, we point out that the detection of backdoors (a training-time attack) is hindered as natural backdoor-like patterns occur even in benign neural networks. The question whether a pattern is benign or malicious then turns into a question of intention, which is hard to tackle. A different kind of complexity is added with the large libraries nowadays in use to implement machine learning. We introduce an attack that alters the library, thereby decreasing the accuracy a user can achieve. In case the user is aware of the attack, however, it is straightforward to defeat. This is not the case for most classical attacks described above. Additional difficulty is added if several attacks are studied at once: we show that even if the model is configured for one attack to be less effective, another attack might perform even better. We conclude by pointing out the necessity of understanding the ML model under attack. On the one hand, as we have seen throughout the examples given here, understanding precedes defenses and attacks. On the other hand, an attack, even a failed one, often yields new insights and knowledge about the algorithm studied.

Background of this Dissertation

This dissertation is based on the papers mentioned in the following. I contributed to all papers as one of the main authors. We review the articles in order of appearance in this thesis.

Michael Backes, Patrick McDaniel and Nicolas Papernot had decided on a joint project extending JSMA to malware data [P1]. To this end, Praveen Manoharan and Kathrin Grosse met with Nicolas Papernot to jointly work on the project. All three implemented the evaluation, with Nicolas implementing JSMA and training, Praveen implementing distillation as a defense and Kathrin implementing feature selection and restrictions. After reviewer feedback, Kathrin extended the paper to encompass which features are changed by the attack. All authors wrote and reviewed the paper.

Michael Backes made contact with Neil Lawrence, leading to a joint project with his Post-Doc Michael T. Smith on the security of Gaussian processes (GP) [P2]. Michael T. Smith contributed the initial attacks on GP, which Kathrin Grosse refined to JSMA/FGSM for GP. David Pfaff and Kathrin Grosse jointly wrote the framework for the experiments. David Pfaff further contributed by proposing the threshold-based defense which was then evaluated by Kathrin Grosse. In the following process of extending the paper, Kathrin Grosse extended the attack further to an optimization-based attack on uncertainty. After a discussion with David Pfaff, Kathrin extended the evaluation to Bayesian neural networks. All authors wrote and reviewed the paper.

During Kathrin Grosse's internship at Ian Molloy's group, Ian Molloy, Youngja Park, Taesung Lee and Kathrin Grosse jointly had the idea to extend randomized smoothing for backdoor detection [P3]. After proof of concept experiments implemented by Kathrin Grosse, the measure was refined by all and fully evaluated by Kathrin Grosse. Taesung Lee and Kathrin Grosse jointly wrote down the formal motivation of the measure. All authors wrote and reviewed the paper.

The paper on adversarial initialization or security of libraries [P4] was the result of a long series of meetings with Dietrich Klakow, Marius Mosbach, Thomas A. Trost and Kathrin Grosse. Kathrin Grosse initially figured out how to deactivate neurons using optimization. This was later extended to to the soft-knockout attacks by Thomas A. Trost. Marius Mosbach later suggested to add the shift parameter, leading to the shift attack. Kathrin Grosse implemented all experiments with the exception of CIFAR networks (by Marius Mosbach). Thomas A. Trost contributed the formal analysis for the attacks. Later on, Kathrin Grosse conducted the Stack Overflow study to support the paper. All authors wrote and reviewed the paper.

The main idea for the fifth work [P5] arose after having written the second work [P2]. Kathrin Grosse wrote the formal part about evasion and intellectual property based attacks. She also implemented all experiments, using the framework written previously with David Pfaff. During the whole project, Michael T. Smith was helpful and always eager to discuss upcoming problems. All authors wrote and reviewed the paper.

The idea for the final paper [P6] originated in an independent project with master student Xaver Fabian. Kathrin Grosse decided to use the shape of lottery tickets as a

baseline to test if the investigated attacks had any effect. When running the experiments and computing a baseline, Kathrin Grosse realized that the structure of winning tickets is variable and implemented all necessary experiments to illustrate this. Receiving feedback from anonymous reviewers, she also extended the work to better understand how the resulting subnetworks differ. All authors wrote and reviewed the paper.

[P1] Grosse, Kathrin et al. Adversarial Examples for Malware Detection. In: *ESORICS*. Springer, 2017. ©Springer 2017, 62–79.

[P2] Grosse, Kathrin et al. The Limitations of Model Uncertainty in Adversarial Settings. *BDL@NeurIPS* (2019).

[P3] Grosse, Kathrin et al. A new measure for overfitting and its implications for backdooring of deep learning. *Under submission* (2020).

[P4] Grosse, Kathrin et al. On the security relevance of initial weights in deep neural networks. In: *ICANN*. Springer, 2020. ©Springer, 2020, 3–14.

[P5] Grosse, Kathrin, Smith, Michael T., and Backes, Michael. Killing Four Birds with one Gaussian Process: The relation between different Test-Time Attacks. In: *ICPR*. IEEE, 2020. ©IEEE 2020.

[P6] Grosse, Kathrin and Backes, Michael. How many winning tickets are there in one DNN? In: *SDM*. SIAM, 2021. ©SIAM 2021.

Further Contributions of the Author

[O1] Hanzlik, Lucjan et al. MLCapsule: Guarded Offline Deployment of Machine Learning as a Service. *TCV@CVPR* (2021).

[O2] Döttling, Nico et al. Metrics and adversarial examples. *Under submission* (2020).

[O3] Smith, Michael T. et al. Adversarial Vulnerability Bounds for Gaussian Process Classification. *ML with Guarantess@NeurIPS* (2019).

[O4] Grosse, Kathrin and Vreeken, Jilles. Summarising Event Sequences using Serial Episodes and an Ontology. *DMNLP@ECML/PKDD* (2017).

Related Bachelor Thesis

[BT1] Fabian, Xaver. An Empirical Study on the Evasion Vulnerabilities of Gaussian Proccesses using Expectation Propagation. *Bachelor Thesis, Saarland University* (2019).

Technical Reports of the Author

[T1] Grosse, Kathrin et al. On the (Statistical) Detection of Adversarial Examples. *arXiv preprint arXiv:1702.06280* (2017).

Acknowledgments

First an foremost, I want to thank my supervisor Michael Backes for giving me this great opportunity. I am grateful for his support, skills and feedback throughout the time of my PhD studies.

I would like to express my gratitude towards my thesis committee members Michael Backes and Mario Fritz for their time and effort spent reviewing this dissertation.

My gratitude extends to everyone who has mentored me academically, including but not limited to Jürgen Dix, Carlos Chesñevar, Tarek Besold, Sabine Janzen, Sebastian Gerling, and Tudor Dumitras. I would also like to thank all my Coauthors for the wonderful time and the helpful instructions. Thanks also go to all my peers for making work such an enjoyable experience. Of my peers, particular thanks go to Pascal Berrang, Sven Bugiel, Sebastian Gerling and Rafael Reischuk, who have provided me the template of this thesis.

Furthermore, I would like to express my grattitude towards my family for their support. In particular, I would like to thank my mother for her open ears and patience, my father for countless book recommendations and the music, and my brother for all the little hints and new perspectives. My grattitude extends to all my uncles, aunts, and cousins. Thank you for patiently answering questions, and also simply for being around. I love you.

I also want to thank my friends Doro, Pablo, Katrin, Ronja, Santi, Caro, Viola, Nelson, Sonja, Christophe, Swheta & Tim, Alena, Herbert, Markus and Hanna for their continuous support and open ears. Special thanks go to my partner Thomas, who is the kindest, smartest and most patient person I know.

Apologies to everyone I might have overlooked—it's my memory, not ingratitude.
(David Mitchell)

Contents

1 Introduction

In the past decades, computational power and data storage have become increasingly affordable, calling for algorithms to make sense of large amounts of data. Machine learning (ML) and data mining (DM) readily provided the needed solutions. In particular deep learning or artificial neural networks are able to scale up to large amounts of data. These networks typically perform classification, e.g. when presented with pictures of cats and dogs, they assign the pictures to their correct class. Beyond that, their applications include for example autonomous driving [14, 94], the game Go [120], large scale image recognition [63] and malware detection [25, 61]. However, such a widespread application of ML and DM raises security concerns[9, 95]. For most attacks, effective provable defenses are not known, leading to arms races [3, 16, 73, 117, 128]. In this thesis, we want to shed light on the questions of why achieving ML security is so hard.

We start with evasion attacks, or adversarial examples. To evade a classifier at test time, a trained classifier is presented with a benign input sample with an added perturbation such that the new sample is misclassified [10, 26, 127]. In the first chapter of this thesis, we deal with the evasion of a classification based malware detector [P1]. The interested reader will notice that the detailed attack formulation is quite similar to the equation of the back-propagation algorithm used in training of deep networks. Furthermore, investigating which features need to be changed for the attack to be successful gives us insights on which features determine class affiliation. This points towards the attacks relying on inherent properties, as opposed to fixable bugs. Afterwards, in the same chapter, we turn to two Bayesian models, Gaussian processes (GP) and Bayesian neural networks (BNN) [P2]. These models do not only provide a classification output, but also give their uncertainty in the output. This Bayesian uncertainty, as we show, can also be manipulated by adversarial examples. Furthermore, transferability is shown: an example from one classifier is often also able to fool a second, unknown classifier [96, P2]. We show that this holds as well for classifiers exhbiting Bayesian uncertainty measures. The fact that the attacker does not even need to know the classifier she wants to fool increases the difficulty for the defender drastically.

In the next chapter, we investigate another attack: backdoors[19, 46, 56]. Backdoors are sets of patterns that the attacker adds to the training data which are linked to one or several particular output classes. A real world example would be a cat-dog classifier that classifies any dog as a cat as long as a sunflower is present in the image. A current hypothesis states that backdoors rely on overfitting [135], e.g. the model has not learned general properties, but specific features from the provided data. Given our measure that quantifies sensitivity and overfitting, we show that in contrast to the existing hypothesis, backdoors rather underfit than overfit. Furthermore, even benign networks that are not altered by an attacker contain backdoor-like patterns[P3]. In other words, neural networks in general will yield stable outputs for one class, as long as particular input features are present in the data. Consequently, it might be impossible to detect or verify that a neural network contains a backdoor, if unbackdoored networks contain similar structures as well.

In the following chapter, we show a different kind of security threat that might emerge when applying deep learning. Given the complexity of today's ML libraries, we show that altering the initialization code for a neural network might have devastating effects [P4]. We name this new training-time attack adversarial initialization. Such an

initialization is theoretically straight-forward to recognize, given that the victim *is aware* of the corresponding threat. If this is not the case, the effects caused by adversarial initialization are overlooked and attributed to other causes. This part illustrates the diversity threats can exhibit—the surroundings of the applied algorithm can be as important as the attack surface of the algorithm itself.

However, even when limiting the study only to the ML model itself, defending can become arbitrarily complex when we allow several attacks at once, as we show in Chapter [P5]. The chapter starts with a formal analysis showing that vulnerability is inherent once the Gaussian process (GP) classifier has learned. We proceed and study the relationship of different attacks targeting intellectual property. Furthermore, GP classifiers allow to configure the decision function curvature. In our experiments, for one individual test-time attack, a seemingly secure configuration can be found. This configuration, however, will be vulnerable to a different attack. Hence, re-configuring the classifier by changing its decision function curvature merely changes vulnerability, as opposed to providing security.

Recapping the previous chapters, we have seen several reasons why ML security is hard. However, we would like to end this thesis on a positive tune. In this chapter, we deal with a slightly different perspective on the hardness of machine learning. To this end, we investigate the lottery ticket hypothesis [34, 36, 41]. This hypothesis focuses on the winning subnetwork that emerges from an iterative pruning process. As a security researcher, one might be curious if this winning ticket can be *altered* such that the victim who trains the model obtains a model that is not optimal anymore. However, the iterative pruning procedure effectively picks a new winning subnetwork in each training run [P6]. The overlap between two runs does not exceed what would be expected if the tickets were chosen randomly. In this sense, the hypothesis cannot be used to derive an attack, but increases our understanding on how ML models differ. This helps us undeerstanding why for example evasion attacks are able to fool several models, yet these models often do not output the same class for the malicious input. Our last isnight is thus how the knowledge in ML and the security of ML benefit from each other.

Structure The remainder of this thesis is structured as follows. In the next chapter, Chapter 2, we provide all necessary background for this thesis. We then review related works in Chapter 3. In Chapter 4, we investigate evasion attacks or adversarial examples. Afterwards, in Chapter 5, we study whether backdoors can be detected in neural networks. We then turn to the security of today's ML libraries in Chapter 6. In Chapter 7, we study the security of an ML model when several test-time attacks are considered at once. We then describe our work concerning the lottery ticket hypothesis in Chapter 8. Finally, in Chapter 9, we draw a conclusion.

2
Background

Overview

In this chapter, we introduce the basic concepts used in this thesis. We start with a formalization of classification, and continue to introduce the classifiers applied. This includes deep neural networks (DNNs), Gaussian processes (GPs), and Bayesian neural networks (BNNs). We then review adversarial machine learning, and conclude the section with a short description of the data-sets used in the individual chapters.

Classification

In the general setting of classification, we are given a labeled set of n training instances. These instances are in the form of feature vectors X, where each individual feature vector x is composed of d features x_i (with $0 < i \leq d$). Each x is associated with a label $y \in \{1, \ldots, c\}$, with c as the number of classes. The goal of classification is to adapt the parameters or weights θ of a *classifier* $f(\_, \theta)$ such that, given further test samples x^*, f predicts $f(x^*, \theta) = y^*_f$ such that $y^*_f = y^*$: the predicted label corresponds to the correct label. It is possible that the classifier *overfits*, and does not generalize properly from the training data. This becomes evident if the training accuracy is high (or the training loss is low), but the test accuracy is low (or the loss is higher) [11]. In other words, instead of fitting only the underlying structure in the data, the classifier also learned irrelevant noise which influences the classification.

In a binary classification problem, the fit of a classifier can be measured using the receiver operating characteristic (ROC). This plot depicts for all possible values of the decision threshold the performance in classification. More concretely, to obtain the ROC-curve, we plot the true positive rate against the false positive rate. The ROC-plot can be condensed into a single number, the area under the curve (AUC).

Before we review more in detail the used classification algorithms, we would like to briefly introduce support vector machines (SVMs). In a nutshell, SVMs pick the most difficult training instances (support vectors) to form the decision boundary.

Deep neural networks

We briefly review deep neural networks (DNN), an algorithm to perform classification. To this end, we first review the algorithm, describe how initial weights are chosen and then detail the lottery ticket hypothesis.

DNN are layered classifiers where input x is propagated through a parametrized layer i, $f_i(x, \theta_i)$. The parameters θ_i are also called weights and biases, and are adapted during training. More specifically, in each layer, the input is multiplied with the weights, the bias added, and then a nonlinear activation function is applied. The resulting output of a layer is then used as input for the following layer, $f_{i+1}((f_i(x, \theta_i), \theta_{i+1})$. The output of the last layer is commonly referred to as logits l. These logits are then fed into a softmax function to obtain a normalized output o_j for logit j (l_j) for the c classes

$$o_j = \frac{e^{l_j}}{\sum_{k \in c} e^{l_k}} \ . \tag{2.1}$$

The individual layers can differ in their structure. Dense layers for example implicitly assume that features are independent. To process image data, for example, convolutional layers are used. In a nutshell, these layers achieve shift and space invariance by sharing a large fraction of their weights. For any type of layer, however, the weights have to be initialized in a good manner to achieve good results in training.

Initialization of deep neural networks

Training and in particular initialization of deep neural networks is still based on heuristics, such as breaking symmetries in the network, and avoiding that gradients vanish or explode [7, 98]. State of the art approaches rely on the idea that, given a random initialization, the variance of weights is particularly important [47, 48] and determines the dynamics of the networks [59, 100]. In accordance with this, weights are nowadays usually simply drawn from some zero-centered (and maybe cut-off) Gaussian distribution with appropriate variance [40], while the biases are often set to a constant. The order of the weights is typically not considered, so an adversarial (or simply unlucky) permutation with particularly bad properties has a good chance of being overseen, if the user is not aware of this kind of problem.

These insights have led to a variety of recipes for initializing DNNs. Before DNNs gained the popularity they have today, rather complex methods for obtaining good initializations were discussed, e.g. [29, 31, 143]. More modern alternative ideas are for example layer-sequential initializations [142]. Modern networks have also grown larger in their number of parameters, raising the question of how to obtain smaller, yet well performing networks. One approach to this problem is the lottery ticket hypothesis.

The lottery ticket hypothesis

A recent trend proposed to prune DNN during training time [30, 34, 41, 88], yielding an iterative process. The network is trained, pruned, and training restarted with the subnetwork resulting from pruning. This procedure is repeated several times, with more and more weights being removed. Many approaches rely on the idea of a winning sub-network that emerges in this process, a so called *winning ticket*. Frankle et al. [34] brought forward the original hypothesis introducing winning tickets. More concretely, Frankle et al. [34] state

> **The Lottery Ticket Hypothesis.** A randomly-initialized, dense neural network contains a sub-network that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

Much effort focuses on tickets for very large neural networks or making training more efficient [23, 38, 41, 134], or studying the hypothesis formally [81].

This concludes the background on DNN.

Bayesian uncertainty

Before we introduce Gaussian process classifiers and Bayesian neural networks, we want to explain in how far they differ from neural networks. The neural network output from

equation 2.1 might give the user the impression that the neural network is able to express how *certain* it is for a given input. However, equation 2.1 is a normalization of the network output, and not a probability. In contrast, Bayesian probability theory allows us to reason about confidence in a mathematically grounded manner. The classification task is then, using Bayes rule, represented as

$$P(y \mid x) = \frac{P(x \mid y)P(y)}{P(x)} \ , \tag{2.2}$$

where $P(y)$ is the distribution of the labels, $P(x)$ is the distribution of the samples, and $P(x \mid y)$ is the distribution of the samples given the labels. This models the probability we are after, $P(y \mid x)$: the probability of a label given a particular sample. Finally, to obtain a classification, we have to apply a decision rule on the probability.

Let us now consider how to obtain a classifier from Equation 2.5. We first note that the denominator, or $P(x)$, is effectively constant, and thus omitted in the following. The nominator is equivalent to a joint probability, we can write

$$P(x \mid y)P(y) = P(x_1, x_2, \dots x_d, y) \ . \tag{2.3}$$

Applying the conditional probability with the chain rule repeatedly, we rewrite as

$$P(x_1 \mid x_2, \dots x_d, y)P(x_2 \mid x_3, \dots x_d, y)P(x_d, y)P(y) \ . \tag{2.4}$$

Under the assumption that the probabilties are mutually independant, the term for each x_i is reduced to $P(x_i \mid y)$. We further rewrite, for a class j,

$$P(y_j) \prod_{i=1}^{d} P(x_i \mid y_j) \ , \tag{2.5}$$

which we compute for all c classes. The class with the largest probability corresponds to the prediction of our *naïve Bayes classifier*. In practice, obtaining these probabilities boils down to counting the frequencies in the data. The notion of naïve for the classifier however stems from the assumption that the feature probabilites are mutually independant.

In practice, it is not justifiable to assume that features are independent. For example in natural images, neighboring pixels tend to have similar values. Analogous reasoning can be made for feature representations of code, language or network traffic. As a consequence, for more complicated and advanced applications, we do not assume that features are independent. Then, however, determining the actual probabilities is computationally very hard [39]. In the following, we introduce two classifiers that approximate these probabilities in different ways.

Gaussian processes

We further use Gaussian Process Classification (GPC) [104] for two classes using the Laplace approximation. The goal is to predict the labels Y_t for the test data points X_t accurately. In GP, we use a covariance function, also called kernel or similarity metric, which we introduce in detail after explaining GPC.

We first introduce Gaussian Process regression (GPR), and assume that the data is produced by a GP and can be represented using a covariance function k:

$$\begin{bmatrix} Y_{\text{tr}} \\ Y_t \end{bmatrix} = \mathcal{N}\left(0, \begin{bmatrix} K_{\text{tr}} & K_{\text{tt}} \\ K_{\text{tt}}^T & K_t \end{bmatrix}\right) , \tag{2.6}$$

where K_{tr} is the covariance of the training data, K_t of the test data, and K_{tt} between test and training data. The transpose is denoted as ^{T}. Having represented the data, we now review how to use this representation for predictions. As we use a Gaussian model, our predictions are Gaussian, too, with a predictive mean and a predictive variance which we define now. At a given test point x', assuming a Gaussian likelihood function, the predictive mean y^* is

$$y_{\text{t}}^* = K_{x'}^T K_{\text{tr}}^{-1} Y_{\text{tr}} , \tag{2.7}$$

where $K_{x'}^T$ is the vector with the distances from x' to each training point.

For brevity, we do not detail the procedure for optimizing the parameters of the covariance function k that defines our Gaussian process. Instead, we outline how to alter this regression model to perform classification. Since our labels Y_t are not real-valued and non-Gaussian class labels, we apply a link function $\sigma(\cdot)$ that normalizes the output to be in range $[0, 1]$. This procedure is called Laplace approximation.

In other words, GP does not learn any explicit weights θ. Instead, we adapt the covariance metric k to fit the training data. For a given test point, k weights all training points with their labels, resulting in the output for this test point.

Covariance functions

We introduce the most common similarity metric in GP, the RBF kernel. This metric k outputs for two points x and x' a similarity defined as

$$k(x, x') = \exp\left(-\frac{l|x - x'|_2^2}{2\sigma^2}\right) , \tag{2.8}$$

where the L_2-distance between two points is rescaled by lengthscale l and variance σ^2. These two parameters, σ^2 and l, form the parameters θ which are fitted or learned during training.

In particular, the lengthscale l affects how local the resulting similarity metric is: a small l yields for example a very local classifier with high decision function curvature. On the other hand, choosing large l yields a classifier with a flat decision function curvature. To affect curvature, l is set before training, and is afterwards not adapted during optimization. Furthermore, l influences how fast the kernel decays. Since we use the exponential function, the output similarity approaches 0 as the distance (rescaled by l) gets larger. This property is called ablation, and is useful for outlier detection or open set tasks [115]. The faster the similarity abates, the more local the classifier and the steeper the decision function.

Other kernels can be used in GP, where not all kernels have an abating property. Other (non-abating) kernels used in this thesis include the linear kernel $k(x', x) = \sigma x' x^T$, where only σ is learned. The polynomial kernel, $k(x', x) = (\alpha x' x^T + c)^d$, contains more learnable parameters: α, c, and d.

Bayesian neural networks

We closely follow the description of Bayesian Neural Networks (BNN) by Smith and Gal [121]. BNN are, analogous to deep neural networks (DNN), functions that consist of layers which are parametrized by θ. In contrast to (non-Bayesian) DNN, these parameters θ of the BNN are not seen as fixed values to be optimized, but treated as random variables. We thus place a prior distribution $p(\omega)$ on the weights. We further estimate the likelihood function $p(y \mid x, \omega)$ that gives the probability of a label y given the input x and the parameter values. To conduct inference, we marginalize over the parameters of the network. In contrast to the GP described in the previous subsection, it is not possible to fully integrate out uncertainty. The uncertainty measures are thus approximated, for example using Variational inference.

Variational inference is a general technique to approximate complex probability distributions. More concretely, we approximate the intractable posterior distribution $p(\omega \mid X, Y)$ with a more tractable, simpler distribution $p_\theta(\omega)$. For neural networks, a typical approximating distribution is dropout, in other words randomly setting some of the units within the network to zero. This dropout distribution is still challenging to marginalize, but it is straight forward to sample from. In this sense, a Monte Carlo estimator serves to apporximate the true posterior $p(\omega \mid X, Y)$ [121].

Adversarial machine learning

In this part, we review all relevant work concerning the security of machine learning. We first describe adversarial examples or evasion, and continue with poisoning or training time attacks. Finally, we review attacks targeting intellectual property like membership inference, model stealing, and model reverse engineering.

Adversarial examples or evasion

Evasion attacks were introduced independently from the security [26, 10] and the ML community [127]. We will use in the following the formalization in [10] to introduce the concept of evasion attacks. Given a trained classifier f, which, for simpicity, solves a binary classification problem for input x. Let us assume that $f > 0.5$ corresponds to output class one, whereas $f \leq 0.5$ corresponds to output class zero. For sample x, which is from class one, we obtain the adversarial perturbation δ by solving

$$x + \delta = \arg\min_{\delta} f(x + \delta) \qquad \text{s.t.} \quad \delta < \delta_{\max} \ , \tag{2.9}$$

where δ is upper bounded by $\delta_{\max}$. Minimizing the output of f will eventually lead to misclassification if feasible given the constraint on δ. However, we might remove the cosntraint on δ and only minimize the confidence or output f return for its classification. This yields the additional advantage that if we tested the adversarial example $x + \delta$ on a second classifier f', it might be misclassified as well. This propoerty of adversarial examples is called trasnferability.

In general, we distinguish targeted and non-targeted attacks. In targeted attacks, the adversary can choose the output class of f. We briefly address how to measure δ.

We might use the L_0-metric which counts the number of changed features. It is well suited for binary data, such as malware features. The L_2-metric is equivalent to the euclidean or squared-root distance, and thus well suited for images. Another metric for images is the L_∞-metric that measures the largest change introduced.

Many algorithms exist to craft adversarial examples. We recap the algorithms used in Chapter 7 and Chapter 4. The fast gradient sign method (**FGSM**) [43] is an untargeted one-step attack. In mathematical terms, the perturbation δ to fool the classifier is computed as

$$\delta = \epsilon \times \text{sign}(\nabla_x F(x, \theta)) \ .$$

In other words, one step adds the gradient of the model's loss w.r.t. the input x' to the original sample. The step size is parametrized by ϵ. FGSM minimizes the L_∞-norm, as the same change is applied to all features. The Jacobian-based saliency map approach (**JSMA**) [97] picks iteratively a pixel for perturbation that maximizes the output for the target class and minimizes the output for all other classes. This search is executed iteratively until misclassification is achieved or the perturbation is too large.

Finally, the $\mathbf{L_x}$ attacks [17] formulate evasion as an iterative optimization problem. The basic L_2 attack is formalized as the following optimization problem

$$\min_{\delta} \parallel 0.5(\tanh(\delta) + 1) + x \parallel_2 + sg(0.5(\tanh(\delta) + 1)) \ ,$$

where tanh ensures the box-constraint to enforce that no feature is set to higher values than in benign data. Term s compromises between the constraint and function g. This function represents how confidently the network f misclassifies $x + \delta$. Other variants of this attack minimize the L_0 or L_∞ norms [17].

While numerous defenses for evasion have been proposed [T1, 13, 15], the attack is far from being mitigated and there is an ongoing arms race [3, 16, 73, 117].

Poisoning

The earliest works to attack training, also called poisoning [8, 109], altered the training data or labels to decrease accuracy for the resulting classifier. At the time, this included methods like support vector machines. Recently, efforts were made to extend poisoning to deep learning. Due to the flexibility of deep models and the difficulties to harm overall security, however, instead **backdoors** were introduced [19, 46, 56]. Backdoors are small patterns that are added to the training data and are linked to one or several target classes during training. An example is a cat-dog classifier. The attacker introduces, to the training data, a couple of images of dogs with sunflowers labeled as cats. If the network, during deployment, is shown a sunflower, it will likely classify this image as a cat. Defending backdoors is an open research problem [18, 78, 132, 135], which has shown to lead to an arms race [128].

Model reverse engineering, model stealing and membership inference

We now review attacks that harm the Intellectual Property (IP) of the model owner, where we focus on GP classification. We start with model reverse engineering, continue with membership inference and finally describe model stealing.

In **model reverse engineering**, given a trained classifier with black-box access, the attacker tries to infer hyper-parameters of the model using specifically crafted queries [92]. For GPs, possible parameters to be targeted are for example the lengthscale(s) and the chosen covariance function.

Membership inference describes an attack which aims to learn whether or not some samples were used to train the model [52, 111, 118]. Such attacks are generally run in a black-box setting, and exploit differences in confidence for trained and unseen data. In contrast to deep learning, a GP is not forced to be overly confident on training data, so these attacks are non-trivial. In our evaluation, we use both confidence (predictive mean) and the predictive variance to deduce this information—a slight variation of known attacks.

A **model stealing** attack aims to reproduce the full black-box model [96, 130]. For GPs, this amounts to finding all parameters learned during training and which training data was used, as this information defines the GP completely. In the case of GPs, this attack is a combination of the previous two attacks.

Data-sets

To conclude, we review all data-sets used in this thesis. We start with the security related data-sets such as malware Data-sets, and then move to other Data-sets like Spam detection, credit admission, and fake banknote detection. Afterwards, we briefly introduce typical vision Data-sets like MNIST, Fashion MNIST, SVHN and CIFAR10.

Hidost The first malware data-set consists of the PDF malware data of the Hidost Toolset project [123]. The data consists of $439,563$ PDF Malware samples, of which $32,567$ are malicious and $407,036$ are benign. Each sample consists of $1,223$ features, where feature vectors are likely to be sparse, and are binary and real-valued. This Data-set is used in Chapter 7.

DREBIN The second data-set (introduced by Arp et al. [2]), contains $123,453$ benign and $5,560$ malicious Android Applications, totaling in $129,013$ instances. Each sample consists of $545,333$ binary malware features. This data-set is part of Chapter 4 and 7. In the latter, however, as the number of features in this data-set is very large, we restrict ourselves to the manifest features, as these are editable, leaving us with $233,655$ binary sparse features.

Spam The third data-set contains $4,601$ samples for Spam detection [72]. The number of features is 57, of which 54 features are continuous and represent word frequencies or character frequencies. The three remaining integer features contain capital run length information. This data-set is slightly imbalanced: roughly 40% of the samples are classified as Spam, the remainder as benign e-mails. We split this data-set randomly and use 30% as test data. This data-set is used in Chapter 7.

Credit The fourth data-set contains 14 features about Australian credit card applications and whether or not they were granted [72]. The features are real, binary, and nominal. There are 690 instances, or applications, 44.5% of which were granted. We split this data-set randomly and use 30% as test data. This data-set is part of Chapter 7.

Bank The fifth data-set contains 4 precomputed features from pictures of banknotes [72]. For each of the $1,372$ instances, we must decide whether the banknotes are real or fake. All features are real valued, with both classes being the same size. We split this data-set randomly and use 30% as test data. This data-set forms part of Chapter 7.

MNIST The sixth data-set is the MNIST benchmark data-set [69]. It consists of 28×28 pixel black and white images of handwritten single digits. There are $50,000$ training and $10,000$ test samples, with all classes roughly the same size. This data-set is used in Chapters 4, 6, and 7. In Chapters 4 and 7, we select a variety of number vs. number tasks, in the latter also number vs. all tasks.

Fashion-MNIST Another typical benchmark is the Fashion-MNIST data-set [139]. Analogous to MNIST, it contains 28×28 pixel black and white images. In contrast to MNIST, the images contain clothing of ten different categories such as shirts, boots, sandals, and bags that are to be classified. There are $60,000$ training and $10,000$ test samples. This data-set is part of Chapters 4, 5, 6, and 8. In Chapter 4, binary subtasks of Fashion-MNIST are used.

SVHN Further, we use the SVHN data-set [90]. It consists of 32×32 colored images of house number digits. There are $73,257$ training and $26,032$ test samples. In Chapter 7, we select a variety of number vs. number tasks.

CIFAR10 Finally, the CIFAR10 data-set [64] consists of 32×32 rgb images with ten classes like truck, plane, horse or deer. Each class has $6,000$ training and $1,000$ test samples. This data-set is used in Chapter 6 and 8.

3
Related work

Overview

In this chapter, we review the related work from all chapters. As some chapters overlap in their related work, the topics are not strictly ordered as they appear in the chapters. We start with evasion, an attack studied in Chapters 4 and 7. Afterwards, we review related work for the training time attacks studied in Chapter 5. We then focus on initialization and attacks on initialization of deep learning, as described in Chapter 6. In the following, we review IP-based attacks on GP and works that study the relationship between several test time attacks, as examined in Chapter 7. To conclude the section, we discuss the related work of Chapter 8 dealing with the lottery ticket hypothesis.

Evasion

In the first part, we review literature about evasion attacks focused on two aspects: malware and Bayesian methods. This corresponds to the related work of the research in Chapter 4 and 7.

Evading malware detectors. Given the automation of malware detection in security [65, 107, 113, 116], evasion of such detectors was investigated early with several works targeting PDF malware detectors [10, 80] and real world systems [66]. The first part of Chapter 4 is an extension of a gradient-based attack changing individual features [97], with the contribution to use this attack against Android malware classification without harming functionality of the original malware. Later, follow-up works focused even more on the ability to change code without harming functionality by transforming the actual malware [62, 99].

Evading GPs. Empirical evasion security has been studied on Gaussian processes (GPs) [P2, 13, 15]. GP also allows one to bound evasion vulnerability [O3, 12]. Our work in Chapter 7 focuses instead on understanding the relationship between decision function curvature and different algorithms for evasion, as well as the broader context of test time attacks.

Evasion and model uncertainty. A more intriguing property of GPs to study in this context is *model uncertainty* in Chapter 4. Bekasov and Murray [6] show the importance of priors in robustness. Furthermore, Bugonovic et al. [13] propose a robust optimization method for GP, which is tested on small datasets only. Also, the authors do not evaluate their approach on adversarial data. Further, Bradshaw et al. [15] investigate Gaussian Hybrid networks, a deep neural network (DNN) where the last layer is replaced by a GP. Melis et al. [84] add a 1-class SVM as a last layer of a DNN to build a defense based on uncertainty. They show that this defense can be circumvented, analogous to our work. In Chapter 4, we go a step further and test whether principled model uncertainty as a defense can be circumvented in a black-box setting.

Evasion and principled model uncertainty. Another line of work focuses on models allowing intrinsic principled uncertainty measures. For example, Gal and Smith [121]

propose an attack to sample garbage examples in the pockets of the uncertainty of Bayesian neural networks (BNNs). BNNs are further investigated by Rawat et al. [105]. They test FGSM adversarial examples on Bayesian networks and find notable differences in model uncertainty for such examples. Li and Gal [71] observe differences for high confidence adversarial examples. Furthermore Smith and Gal [121] conclude that Mutual Information of an ensemble of Bayesian networks detects adversarial data. In the second part of Chapter 4, we propose adversarial examples optimized also on intrinsic model uncertainty and show their transferability. This contradicts previous claims that principled model uncertainty is more robust or more difficult to fool.

Transferability and Bayesian uncertainty. In the first chapter, we show that *transferability* also holds for model uncertainty. General transferability was first shown by Papernot et al. [96]. Rozsa et al. [108] study transferability for different DNN architectures. Further Liu et al. [76] show that using an several models to compute the examples to be transferred improves the success of targeted transferred examples. Demontis et al. [28] additionally reason that gradient alignment and mode complexity affect the success of transferability. To the best of our knowledge, there exist no works so far studying the transferability of adversarial examples across different models enabling principled uncertainty measures.

Evasion and curvature. In Chapter 7, we investigate the relationship between curvature and evasion attacks. Such a relationship has been confirmed in linear models like support vector machines [110] and used for mitigations in deep neural networks [51, 102]. To the best of our knowledge, there are no works studying the relationship between curvature and vulnerability in GPs.

Formal work on evasion. Formal works in the area of evasion have for example derived secure defenses via certificates, for example in randomized smoothing [22, 70]; or by bounding the changes [O3, 12] an attacker can introduce. However, our formal analysis in Chapter 7 is unrelated to defenses. More related is a line of works deriving impossibilities using cryptographic primitives [27, O2]. Although our result is close to an impossibility, it does not rely on cryptographic primitives. Cullina et al. [24] present an analysis of evasion on the PAC framework, and how the VC dimension changes if an adversary is present. Tanay and Griffin [129] analyze adversarial examples using their projection onto the decision boundary. In particular, they link a special kind of adversarial examples to overfitting. Our analysis in Chapter 7 instead shows that a secure classifier will become insecure as it learns, e.g. long before overfitting might occur. Our formal evasion analysis for GP is in the finite sample setting. Wang et al. [136] instead give an analysis in the infinite sample limit on the k-nearest-neighbors classifier.

Poisoning, sensitivity, and overfitting

We now review literature about backdoors and other training time attacks. As our approach in Chapter 5 relies on a measure to quantify overfitting or sensitivity, we start with related work in the area of overfitting and sensitivity analysis. Afterwards, we

discuss training time attacks. As the attack in Chapter 6 alters the initial weights and thereby affects training, we also discuss related work in this area.

Measuring overfitting and sensitivity. Jiang et al. [57] give a detailed overview about both empirical and theoretical measures. Here, we review works that are closest to our approach in Chapter 5. Werpachowski et al. [137] propose an overfitting measure based on a large batch of adversarial examples and a statistical test. Ebrahimi et al. [32] measure the generalization gap, e.g. the difference between training and test data using the distance to the margin. Both approaches are similar in that they consider the margin essential - we instead measure sensitivity using Gaussian noise, independent of the decision boundary. Finally, Novak et al. [91] introduce a measure for **input sensitivity**, based on the norm of the input/output Jacobian. They connect their norm to overfitting as well. Further Shu et al. [119] show that test and training data show slightly different fit under input and/or weight perturbations. Chapter 5 differs in two aspects from these previous works. On the one hand, our measure is directly computed on the strongest output class and does not rely on the Jacobian. On the other hand, our work also differs in that it studies the relationship between input sensitivity, overfitting and security of deep neural networks.

Bias-variance-trade-off. Our measure in Chapter 5 is motivated by the bias-variance-trade-off. This trade-off decomposes the expected generalization loss of a classifier into variance, bias, and an irreducible error term [60]. The underlying idea is to compare a classifier across different datasets, also called *random design*. Ba et al. [4] assume instead one dataset, but that the data is noised. This is called *fixed design*, and although we noise the input data, not the labels, this is the setting our motivation stems from. Formerly, the bias-variance-trade-off implied that bias decreases with model complexity, whereas variance increases. Mei and Montaneri [83] and Yang et al. [144] challenge this view: They find evidence that the variance term is instead u-shaped, and decreases in regimes of higher model complexity. In Chapter 5 we provide an orthogonal view, stating that we can measure overfitting locally.

Backdoor detection. Defending backdoors is an open research problem [18, 78, 132, 135], which has led to an arms race [128]. Our backdoor detection mechanism wrongly classifies all candidates generated by Wang et al. [135] as backdoors. This wrong attribution implies the existence of benign backdoors in any network, showing the difficulty of defending against such attacks. We are unaware of another work in that direction. Loosely related is also the work of Wang et el. [135], stating that backdoors rely on overfitting. We revisit this hypothesis and present evidence that backdoors instead rely on underfitting.

Adversarial initialization. Chapter 6 introduces a new training time attack. In contrast, Cheney et al. [20] investigated adversarial weight perturbations at test time (not at training time of the initial weights). Benign hardware failures during training, or weight changes, have been studied as well [133].

Additionally and independently from our work, the security of deep learning frameworks has been investigated. This includes security relevant bugs in ML frameworks [125], as opposed to our active attacker. Active attackers in such frameworks have been studied, however. Xiao et al. [140] investigate an active attacker that manipulates the image that is passed to the networks at test time. The same authors [141] investigate in how far the loading of the model can be manipulated by an adversary. Further, they investigate an attacker who alters the images fed into the model at training time. In contrast, in Chapter 6 our attacker manipulates the *initial model*, and does *not* assume any knowledge about the training data.

Closest to Chapter 6 are Liu et al. [74], who target the weights of an SGD-trained model which consecutively overfits the data. There are several differences to our contribution: (1) our attacks are independent of the optimizer and other hyper-parameters, and (2) the damage of decreased accuracy is more severe than overfitting. Furthermore, (3) our attack is also more stealthy, as the statistics of the original weights are preserved, and (4) our attacks take place *before* training.

GP and different test time attacks

To the best of our knowledge, few works have studied the relationship between different attacks. Most works focus on deep learning, and on at most two attacks. For example Suciu et al. [126] study evasion and training time attacks jointly. Song and Mittal [79] show that neural networks that are robust against evasion are more vulnerable to membership inference. Along these lines, there are defenses taking into account several attacks on deep learning [21, 58]. We do not study defenses, as we are aware that GPs are vulnerable [P2]. We instead focus on an in depth study of the *relationship* between several attacks, and are unaware of any similar work.

GP and membership inference. Concerning membership inference, most works have been carried out in the context of deep learning or machine learning as a service, e.g. black-box models [111, 112, 118]. In Chapter 7, we specifically study GPs with a focus on the relationship to other test time attacks. A formal approach concerning GPs and membership inference is the recent work on differential privacy for GP (see for example [122]). In Chapter 7, we do not study formal guarantees but instead the relationships across different test time attacks.

GP and model stealing. Analogously, many works in the area of model stealing have been agnostic about the deployed model or focused on deep learning [96, 130]. Other works study model stealing indirectly by deriving the amount of queries needed to evade a black-box classifier [89, 124]. None of these works have attempted to link model stealing to other test time attacks as done in Chapter 7.

GP and model reverse engineering. The only paper we are aware of introducing model reverse engineering [92] is strictly focused on neural networks, predicting features like the usage of dropout. We are thus the first to show similar attacks on GPs' lengthscale and on the kernel used.

The lottery ticket hypothesis

Frankle et al. [34] introduced winning tickets. Follow-up work, for example Gondara et al. [42], use tickets to obtain differentially private neural networks. Many works focus on tickets for very large neural networks or training efficiency [23, 38, 41, 134]. Along these lines, You et al. [146] presented evidence that winning tickets emerge early in training. In Chapter 8, we offer a different perspective on winning tickets, as we study how many tickets emerge from a fixed initial set of weights.

Closest to Chapter 8 is Frankle et al. [36] who study the effect of SGD noise on winning tickets. We instead investigate how much resulting networks differ without rewinding in terms of their masks when randomness is not fixed or partially fixed. On the one hand we confirm results from Frankle et al. [36] about early training stages where networks diverge, however we show that divergence is less if stochastic elements of training are removed. We also study divergence differently, as we consider the distance between masks and whether networks at the end of training are functionally equivalent.

Orthogonal work by Ramanujan et al. [103] shows that winning sub-networks can be distilled from a network without any training. In this case, the structure is learned.

4
Evasion

Introduction

In evasion, a trained classifier F is presented with a benign sample x with an added perturbation such that $F(x) \neq F(x + \delta)$. The input $x + \delta$ is also called adversarial example. In this section, we first have a look at an attack targeting malware classifier with discrete features [P1]. After investigating which specific malware features are changed to achieve missclassification, we turn to two Bayesian models, Gaussian processes (GP) and Bayesian neural networks (BNN). We show how an evasion attack can be extended to not only fool classification but also Bayesian uncertainty estimates of a model. Furthermore, transferability is shown: the adversarial example crafted for one model is often also able to fool a second, unknown classifier [P2].

Threat model. We specify the different adversaries for the two subsection using the FAIL [126] model. F denotes the attacker's knowledge about the features. A denotes knowledge about the algorithm applied and I about the training data. L summarizes whether changes to the data by the attacker are constrained.

In both settings, the attacker knows the features (F) and the algorithm applied (A). In the second setting, however, the attacker transfers adversarial examples from GP to a BNN, and is unaware that this particular algorithm is applied. In both cases, the attacker has information about the training data (I). This is subtle in the second part: both GP classification (GPC) and BNN are trained on the same training data. In the first, the malware case, the feature changes by the attacker are constrained. In the second case, no constraints are taken into account by the attacker (L).

Adversarial examples for malware detection

Our goal is to have a malicious application classified as benign, i.e. given a malicious input x, the classification outputs benign. Note that our approach naturally extends to the symmetric case of misclassifying a benign application.

We adopt the adversarial example crafting algorithm based on the Jacobian matrix

$$JF = \frac{\partial F(x)}{\partial x} = \left[\frac{\partial F_i(x)}{\partial x_j}\right]_{i \in 0,1, j \in [1,m]}$$

of the neural network F put forward by Papernot et al. [97]. Despite it originally being defined for images, we show that a careful adaptation to a different domain is possible.

To craft an adversarial example, we take two steps. At first, we compute the gradient of F with respect to x to estimate the direction in which a perturbation in x would change F's output. In the second step, we choose a perturbation δ of x with maximal positive gradient into our target class y'. For malware missclassification, this means that we choose the index $i = \text{argmax}_{j \in [1,m], x_j = 0} F_0(\mathrm{x}_j)$ that maximizes the change into our target class 0 by changing x_i. We repeat this process until either a) we reached the limit for maximum amount of allowed changes or b) we successfully cause a misclassification.

Classifier/MR	Accuracy	FNR	FPR	MR	Dist.
Sayfullina et al. [114]	91%	0.1	17.9	–	–
Arp et al. [2]	93.9%	1	6.1	–	–
Zhu et al. [149]	98.7%	7.5	1	–	–
ours, 0.3	98.35%	9.73	1.29	63.08	14.52
ours, 0.4	96.6%	8.13	3.19	64.01	14.84
ours, 0.5	95.93%	6.37	3.96	69.35	13.47

Table 4.1: Performance of classifiers. Given are malware ratio (MWR), accuracy, false negative rate (FNR) and false positive rate (FPR). The misclassification rates (MR) and required average distortion (Dist. in number of added features) with a threshold of 20 modifications are given as well. The lower five approaches use the DREBIN data set.

Restrictions on adversarial examples. Note finally that we only consider positive changes for positions j at which $x_j = 0$, which correspond to adding features the application represented by x (since x is a binary indicator vector). We discuss this choice in the next subsection.

To make sure that modifications caused by the above algorithms do not change the application too much, we bound the maximum distortion δ applied to the original sample. As in the computer vision case, we only allow distortions δ with $0 < \delta < k$. We differ, however, in the norm that we apply: in computer vision, the L_∞-norm is often used to bound the maximum change. In our case, each modification to an entry will always change its value by exactly 1, and we thus use the L_1-norm to bound the overall number of features modified. We further bound the number of features to $k = 20$.

While the main goal of adversarial example crafting is to achieve misclassification, for malware detection, this cannot happen at the cost of the application's functionality: feature changes can cause the application in question to lose its malicious functionality. Additionally, inter-dependencies between features can cause a single line of code that is added to a malware sample to change several features at the same time.

To maintain the functionality of the adversarial example, we restrict the adversarial crafting algorithm as follows: first, we will only change features that result in a single line of code that needs to be added to the real application. Second, we only modify *manifest* features which relate to the `AndroidManifest.xml` file contained in any Android application. Together, both of these restrictions ensure that the original functionality of the application is preserved. Note that this approach only makes the crafting adversarial examples harder: instead of using features that have a high impact on misclassification, we skip those that are not manifest features.

Empirical setting. Since the binary indicator vector X we use to represent an application does not possess any particular structural properties or interdependencies, like for example images, we apply a regular, feed-forward neural network. We use a rectifier as the activation function, and to train our network, we use standard gradient descent and standard dropout. We train numerous neural network architecture variants. Since the DREBIN dataset has a fairly unbalanced ratio between malware and benign applications, we experiment with different ratios of malware in each training batch to compare the achieved performance values. The number of training iterations is

then set in such a way that all malware samples are at least used once. We evaluate the classification performance of each of these networks using accuracy, false negative and false positive rates as performance measures. We decided to pick an architecture consisting of two hidden layers each consisting of 200 neurons and provide more details about the performance of other architecture is a longer version of this paper. In Table 4.1 the accuracy as well as false positive and false negative rates are displayed.

In comparison, Arp et al. [2] achieve a 6.1% false negative rate at a 1% false positive rate. Sayfullina et al. [114] even achieve a 0.1% false negative rate, however at the cost of 17.9% false positives. Saxe & Berlin [113] report 95.2% accuracy given 0.1 false positive rate, where the false negative rate is not reported. Zhu et al. [149], finally, applied feature selection and decision trees and achieved 1% false positives and 7.5% false negatives. As we can see, our networks are close to this trade-offs and can thus be considered comparable to state-of-the-art.

Empirical results. Next, we apply the adversarial example crafting algorithm and observe how often the adversarial inputs are able to successfully mislead our neural network based classifiers. We quantify the performance of our algorithm through the achieved misclassification rate, which measures the amount of previously correctly classified malware that is misclassified after the adversarial example crafting. In addition, we also measure the average number of modifications required to achieve misclassification to assess which architecture provided a harder time being mislead. We allow at most 20 modification to any of the malware applications.

We achieve misclassification rates from roughly 63% up to 69% for the three different malware ratios 0.3, 0.4, and 0.5. The malware ratio used in the training batches is correlated to the misclassification rate: a higher malware ratio generally results in a lower misclassification rate.

While the sets of frequently modified features across all malware samples differ slightly, we observe trends for frequently modified features across all networks. For the networks of all malware ratios, the most frequently modified features are permissions, which are modified in roughly 30-45% of the cases. Intents and activities come in at second place, modified in 10-20% of the cases.

More in detail, the feature `intent.category.DEFAULT` was added to 86.4% of the malware samples when training with malware ratio 0.3. In networks with other malware ratios, the most modified feature was `permission.MODIFY_AUDIO_SETTINGS` (82.7% for malware ratio 0.4 and 87% for malware ratio 0.5).

Other frequently modified features are for example `activity.SplashScreen`, `android.appwidget.provider` or the GPS feature. And while for all networks the `service_receiver` feature was added to many malware samples, other are specific to the networks: for malware ratio 0.3 it is the `BootReceiver`, for 0.4 the `AlarmReceiver` and for 0.5 the `Monitor`.

Overall, of all features that we decided to modify (i.e. the features in the manifest), only 0.0004%, or 89, are used to mislead the classifier. Of this very small set of features, roughly a quarter occurs in more than $1,000$ adversarially crafted examples. A more detailed breakdown can be found in Table 4.2.

	malware ratio 0.3		malware ratio 0.4		malware ratio 0.5	
Feature	total	$> 1k$ Apps	total	$> 1k$ Apps	total	$> 1k$ Apps
Activity	16	3	14	5	14	2
Feature	10	1	10	3	9	3
Intent	18	7	19	5	15	5
Permission	44	11	38	10	29	10
Provider	2	1	2	1	2	1
Service_receiver	8	1	6	1	8	1
$\sum$	99	25	90	26	78	23

Table 4.2: Feature classes from the manifest and how they were used to provoke misclassification. We denote the total number of cases and the number of cases that occurs in more than $> 1,000$ Apps.

Summary and conclusion

As expected, the malware classifier can easily be mislead by the crafted adversarial examples. In these examples, few features are altered. Some of these features are generally used, others depend on the specific network and malware ratio used in training. More in detail, these are the features that were learned in training, or observed to have a large correlation with one class. Also mathematically, computing the gradient not for the features but for the weights yields the training update. We conclude that security in machine learning stems from inherent properties of the classifier, as opposed to fixable bugs.

In the following, we have a look at at different kind of classifier, Gaussian processes (GPs) and Bayesian neural networks (BNNs), both exhibiting uncertainty measures. As we will see, even these principled measures are not immune to adversarial examples.

Bayesian uncertainty and confidence

In the second paper of this thesis [P2], we confirmed previous findings [71, 105] that conventional attacks often lead to noticeable deviations in confidence and uncertainty. Hence, we also adapt the optimization of adversarial examples to account for confidence and uncertainty, thereby introducing *high-confidence-low-uncertainty* (HCLU) adversarial examples. We formalize the computation of HCLU examples as

$$\begin{aligned} \min_{\delta} \quad & \| \delta \|_2 \\ \text{s.t.} \quad & \text{mean}(f(x+\delta)) > 0.95, \\ & \text{var}(f(x+\delta)) \leq \text{var}(f(x)). \end{aligned}$$

where we minimize the perturbation δ using the L_2-norm. An extension to other norms (as in [17]), or a restriction to change only a subset of the features, is however straight forward. We study the L_2-norm as it is differentiable and thus allows to formulate a worst case attacker. Concerning confidence, we demand explicitly that

	Spam	F. MNIST19	F. MNIST57	MNIST19	MNIST38
$\| \delta_{HC} \|_2$	0.006±0.01	0.194±0.036	0.019±0.012	0.053±0.014	0.029±0.011
$\| \delta_{HCLU} \|_2$	0.008±0.006	0.194±0.036	0.019±0.013	0.053±0.014	0.03±0.012
diff. unc.	2.35	0.0	0.89	0.31	0.74

Table 4.3: Average perturbation and uncertainty change of HC/HCLU to benign data.

the resulting adversarial example is confidently classified (first constraint). This first constraint can be parametrized, and any desired confidence can be targeted. Additionally, we require that the uncertainty GPC outputs for the example is as least as low as for the benign counterpart (second constraint). We also specify box constraints for each feature to prevent features set to values outside the usual range.

In the following, we first describe the resulting HCLU examples. Afterwards, we test their transferability to BNN. We conclude the section by summarizing our results.

Properties of HCLU examples. We show the HCLU with the smallest δ in the fifth row of Figure 4.1. These examples are still adversarial: we see in the figure that almost all are visually similar to their benign origin. To verify for all depicted examples that they were altered in the crafting process, we plot the original sample beneath the examples. The success rates on GPC are 100%, where however often the specified confidence of 0.95 is barely not met, and the resulting confidence is around 0.948. Table 4.3 shows the statistics of the adversarial perturbations per feature measures using the L_2-norm. We observe very small changes on spam, large changes on Fashion MNIST19 and descent changes for all other datasets.

We can also craft examples that only maximize confidence by removing the second constraint. Surprisingly, the perturbation δ_{hc} is barely different from the original examples, as visible in Table 4.3: only on the spam data and two MNIST tasks, a difference is observable. The pictures in the fourth row of Figure 4.1 reveal that albeit looking very similar, the examples are actually different. Some features are generally changed, whereas others seem only correlated with uncertainty. We conclude that slightly different features are learned for confidence and uncertainty, respectively. Consequently, for all datasets except Fashion MNIST ankle boot vs trousers, the observed uncertainty is indeed lower when targeted. Concerning the unchanged Fashion MNIST task, we observe that the change in uncertainty for those is examples is only 1% from the original value. In this case, confidence and uncertainty seem to rely on the same features.

Transferability of uncertainty. We now test the effect of HCLU examples on Bayesian neural network's (BNN) uncertainty measures. For misclassification, e.g. non-Bayesian decision boundaries, [96] showed that adversarial examples often transfer as the model's decision boundaries are sufficiently similar. In this experiment, we are interested whether behavior *differs* between benign and adversarial data. We chose Carlini and Wagner's L_2 attack as a baseline: L_2, as in our case, allows the best optimization. We further configure the attack as to increase transferability of the examples (corresponding to

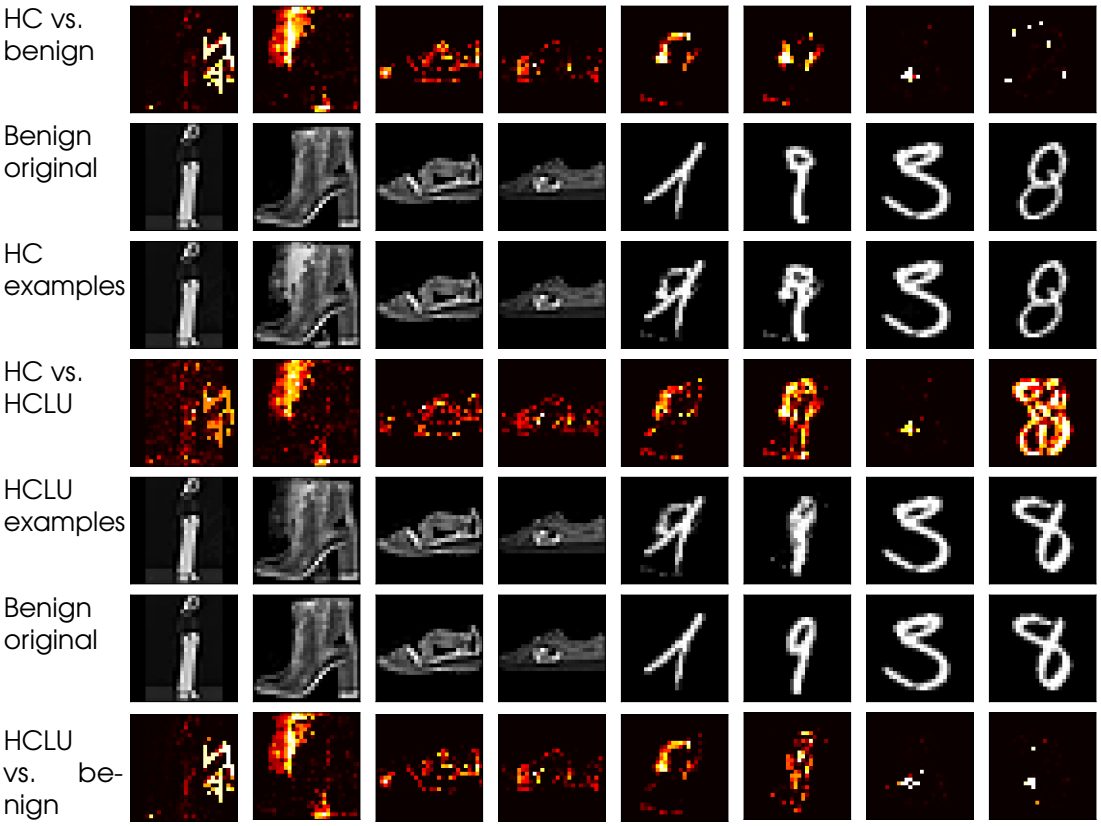


Figure 4.1: Comparison of HCLU, HC, and benign originals. Grey are examples and samples, red plots (row one, four and seven) show differences between images. HC vs. HCLU differences are scaled according to column label. For Fashion MNIST, these scales are also used for the comparisons in row one and seven. We plot all differences using a logarithmic spectrum that allows to see small changes. In the spectrum, colors go from black over red to yellow to white (strongest change). Benign originals are the samples started with to craft the upper HCLU or lower HC example. Figures with differences are best seen in color.

higher "confidence" on the target DNN)[1].

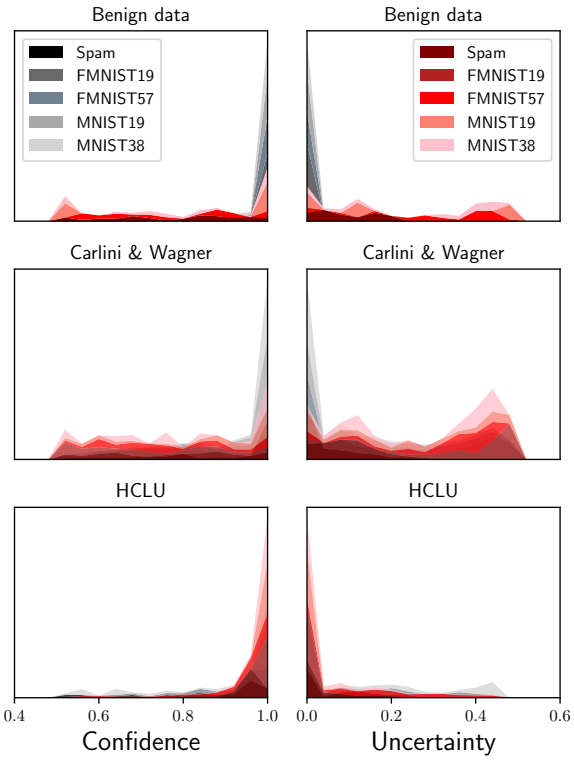


Figure 4.2: Transferability of HCLU examples (bottom) to Bayesian Neural Networks. We consider Carlini & Wagner's L_2 attack as a comparison (middle). Benign data is also depicted as a baseline (top). Correctly classified data is plotted in gray shades, misclassified data in red shades. Figure is best seen in color.

The accuracy on these L_2 adversarial examples on DNN, GPC, and BNN is higher than on HCLU: the average accuracy is between 50% and in some cases higher than 90%. Yet, the accuracy under the transferred attack is always lower than the accuracy on clean test data.

We depict the results concerning Bayesian confidence and uncertainty in Figure 4.2, where we distinguish correctly classified (gray shades) and wrongly classified (red shades) benign and adversarial data. We measure the mean (confidence, left plots) and variance (uncertainty, right plots) of the sampled posteriors and bin them using 25 bins between 0.0 and 1.0. As large parts of the histograms are empty, we plot only the relevant parts. To outline overall trends, we plot the normalized bins of correct and wrongly classified data stacked on top of each other.

In general, the BNN is more confident on benign data/Carlini and Wagner examples that are correctly classified. This observation holds across all data sets. For HCLU, this trend is reversed: the BNN is confident on many misclassified examples. Intriguingly, the BNN outputs low confidence on some HCLU examples which are not misclassified, or correctly assigned to their original class. Analogously, the uncertainty measures are similar between benign data and Carlini and Wagner's attack. Uncertainty is generally low for correctly classified data and high for wrongly classified data. This observations are again reversed for HCLU examples: here uncertainty is often low if an example is wrongly classified.

Summary and conclusion

The features used for confidence and uncertainty are not disjoint. High confidence adversarial examples crafted using a confidence that is not Bayesian, however, are not classified with high uncertainty by GPC or BNN. Adversarial examples based on

[1]We set $\kappa = 0.7$. The attack definition, in a nutshell, defines $\kappa > 0$ to encourage the solver to find a confidently classified example. For details see [17] .

Bayesian confidence, however, do transfer from GPC to BNN. This implies that even though the approximation of the probabilities is quite different, both are, in practice, quite similar.

Conclusion

Evasion is inherent to classifiers, as the gradients of their surface can be used to determine features influencing classification outcome. Consequently, evasion on undefended classifiers is generally very successful. In the second part, we investiagted whether an adversarial examples with high unvcertainty on one classifier, here a GP, is also misclassified with high confidence on a second classifier, here a BNN. We find this to be true. One reason for this transferability could be that although the classifiers are different, they are trained on the same data, and learn similar features. Yet, more work is needed to understand the phenomenon of transferability better, in particular as transferability often harms the efficiency of defenses [3]. In this sense, this chapter shows that in particular Bayesian uncertainty should not be used as a defense.

5
Backdoors

Introduction

In the previous chapter, we have seen that evasion attacks are inherent to classifiers. In this chapter, based on [P3], we investigate another class of attacks, backdoors. To this end, we introduce a measure that quantifies sensitivity and overfitting, W. We show that even networks which are not trained by an attacker contain backdoor like patterns. In other words, neural networks in general will yield stable outputs for one class, as long as particular input features are present in the data.

We start the section by introducing our measure, W. We continue with several experiments supporting that W quantifies overfitting, and then present our results in the context of backdoor detection.

Local wobbliness measure W

We first give a high level intuition of the W-measure, before formally introducing it.

High level intuition. In Figure 5.1, we depict data points of different classes (in different blue shades) with their sampled areas (gray circles). On the left hand side, we show a wobbly, or overfitted classifier. On the right hand side, we observe a good, non-overfitted classifier. The picture illustrates W: at an appropriately chosen radius, overfitting becomes evident as the sampled ball around a test point is not consistently classified. We thus quantify overfitting in the input space by measuring the output space in the area around each test point. Note that this measurement is local, and it can vary across the input space.

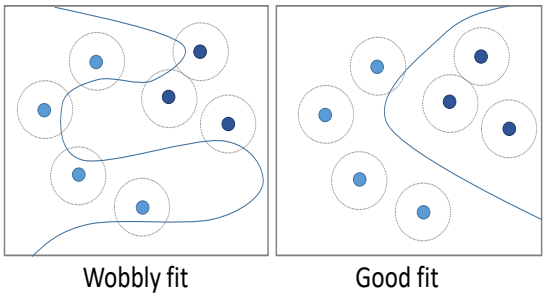


Figure 5.1: Measuring overfitting with W: We sample points (gray circles) around test data (blue dots) to quantify the wobbliness of the decision function (blue line).

Definition of W. An ML classifier $F(\mathbf{x}, \theta)$ trained on i.i.d. data is given. We denote training points as $\mathbf{x}$ and the corresponding label as y. After learning the parameters θ on $(\mathbf{x}, \mathsf{y})$ pairs, the loss L should be minimal. We finally define $\mathbf{x}'$ as a randomly drawn point around $\mathbf{x}$; $\mathbf{x}' \sim \mathcal{N}(\mathbf{x}, \sigma^2)$.

Given this formalization, we use the bias-variance decomposition to divide the loss L into bias and variance as introduced by [60]. We start our analysis with the local measurement of the cross-entropy loss function H, the most common loss function for DNN classification, around $\mathbf{x}$ written

$$L(\mathbf{x}, \mathsf{y}) = \mathbb{E}_{\mathbf{x}'} H(\mathsf{y}, F(\mathbf{x}')) = -\mathbb{E}_{\mathbf{x}'} \sum_i \mathsf{y}_i \log F(\mathbf{x}')_i \quad , \tag{5.1}$$

where y_i and $F(\mathbf{x}')_i$ indicate the respective i-th dimension element, and $\mathbb{E}_{\mathbf{x}'}$ is the mean over data points from the Gaussian distribution $\mathcal{N}(\mathbf{x}, \sigma^2)$. We decompose the loss in

bias and variance by adapting [54] on cross entropy, yielding for $L(\mathbf{x}, \mathsf{y})$:

$$\begin{aligned} &= \mathbb{E}_{\mathbf{x}'}\left[H(\mathsf{y}, F(\mathbf{x}')) - H(F(\mathbf{x}'), \mathbb{E}_{\mathbf{x}'}F(\mathbf{x}')) - H(\mathsf{y}, \mathbb{E}_{\mathbf{x}'}\mathsf{y}) + H(F(\mathbf{x}'), \mathbb{E}_{\mathbf{x}'}(F(\mathbf{x}'))) + H(\mathsf{y}, \mathbb{E}_{\mathbf{x}'}\mathsf{y})\right] \\ &= \underbrace{\mathbb{E}_{\mathbf{x}'}\left[H(\mathsf{y}, F(\mathbf{x}')) - H(F(\mathbf{x}'), \mathbb{E}_{\mathbf{x}'}F(\mathbf{x}')) - H(\mathsf{y}, \mathbb{E}_{\mathbf{x}'}\mathsf{y})\right]}_{\text{bias}^2(\mathsf{y})} + \underbrace{H(\mathbb{E}_{\mathbf{x}'}F(\mathbf{x}'))}_{\text{Var}(F(\mathbf{x}'))} + \underbrace{H(\mathbb{E}_{\mathbf{x}'}\mathsf{y})}_{\text{Var}(\mathsf{y})} \quad . \end{aligned} \tag{5.2}$$

The part we are interested in is $\text{Var}(F(\mathbf{x}')) = -\sum_i \mathbb{E}_{\mathbf{x}'}F(\mathbf{x}') \log \mathbb{E}_{\mathbf{x}'}(F(\mathbf{x}'))$. This measures how consistent the prediction is around $\mathbf{x}$ as we discussed in the high level idea section.

Finally, we define our measure W_e ($\mathbf{W}$ for cross-**e**ntropy loss) as the approximation to $\text{Var}(F(\mathbf{x}'))$:

$$W_e(\mathbf{x}) = \sum_i -A(\mathbf{x})_i log\left(A(\mathbf{x})_i + c\right) \quad , \tag{5.3}$$

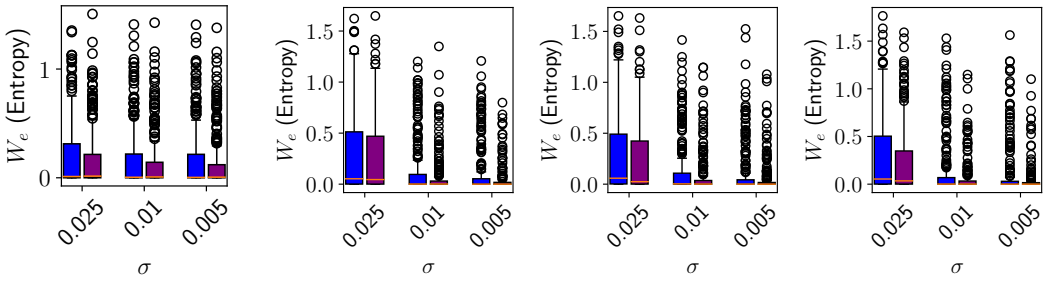
where $A(\mathbf{x})_i = \mathbb{E}_{\mathbf{x}'}(\text{argmax}F(\mathbf{x}') = i)$, the mean of one-hot vectors to use only the top-1 class and c is a small constant (e^{-5}) to avoid computing the logarithm of zero. Using the top-1 class makes it easier to interpret in terms of causing inputs, and enable us to compute this measure even for a black-box model returning only top-1 class. To obtain the measurements, instead of using just one $\mathbf{x}$, we use several data points $\{x^1, \ldots, x^n\}$ and compute the measure for each, hence $W_e = \{W_e(\mathbf{x}^1), \ldots, W_e(\mathbf{x}^n)\}$.

Note again that we have a few critical differences to [60] and [54]. We compute the expectation around $\mathbf{x}$ by drawing random variable $\mathbf{x}'$ from $\mathcal{N}(\mathbf{x}, \sigma^2)$ to obtain local measurements. This is important for two reasons. First, overfitting can be local, and the existing global measures can overlook regional overfitting. Second, global variance can be perceived much higher as it utilizes only a single mean value. Oftentimes, due to the diversity of the input points, using the mean of all points might not be suitable, as it can be far from every data point. Also, we measure many such points to build a distribution where we can apply diverse aggregation or statistical tests.

In other ML models using mean squared error loss, we have $W_v = \mathbb{E}_{\mathbf{x}'}(F(\mathbf{x}) - \mathbb{E}_{\mathbf{x}'}F(\mathbf{x}))^2$. This version shows mostly similar results to W_e, so we will focus on W_e. We also want to briefly remark that [144] decompose the cross entropy function into a sum of Kullback–Leibler-divergences instead of cross-entropies. [54] states that there can be multiple decompositions, especially for an asymmetric loss. Also, the Kullback–Leibler-divergence can have similar trend since KL-divergence is cross-entropy minus entropy.

Empirical evaluation

Unfortunately, W_e comes with many parameter choices. This includes the amount of noised points n, the variance of these points, σ, and the number of test points. A detailed ablation study for all parameters can be found in a long version of [P3]. However, as we show in this section, reasonable values can be found for each. In this section, we vary the amount of noised points n between 500 and 2000, merely to show that there is little effect of this number on W_e as long as it is decently large, namely $W_e > 250$. The choice of σ and the number of test points used to compute the measure are more important. The first, σ, is varied between 0.15, 0.1 as low as 0.005. Small values are appropriate for fine-grained task (distinguish test and training data), larger for more rough-grained



(a) Fashion MNIST. **(b)** CIFAR10 - 3 random draws.

Figure 5.2: Differences of test (blue) and training (purple) data after training using W_e.

(monitor progress during training). Finally, the amount of test points depends on the setting as well. We show that starting with 250 test-points, the measure is fairly stable.

We first describe the overall setting, and then show that W_e is able to distinguish training and test data. Afterwards, we investigate how the measure captures small differences in the training setting, and conclude the section with a detailed study on the measure on individual classes.

Setting. To test W_e, we deploy small networks on Fashion MNIST, where we achieve an accuracy of around 88%. These networks contain a convolution layer with 32 3×3 filters, a max-pooling layer of 2×2, another convolution layer with 12 3×3 filter, a dense layer with 50 neurons, and a softmax layer with 10 neurons. We further experiment on CIFAR10, where we train a ResNet18 [50] 200 epochs to achieve an accuracy of 91.8%.

Plots. We plot the distribution of W_e over n test points using box-plots. These plots depict the mean (orange line), the quartiles (blue boxes, whiskers) and outliers (dots). We follow the standard definition for outliers in [37]: An outlier is defined as a point further away than 1.5 the interquartile range from the quartiles. More concretely, Q_{25} is the first quartile and Q_{75} is the third quartile (and Q_{50} is the median). Value ν is an outlier iff

$$\nu > Q_{75} + 1.5 \times (Q_{75} - Q_{25}) \text{ or } \nu < Q_{25} - 1.5 \times (Q_{75} - Q_{25}) \ , \tag{5.4}$$

in other words if ν is more than 1.5 times the interquartile range $(Q_{25} - Q_{75})$ away from either quartile Q_{25} or quartile Q_{75}.

Measuring overfitting: training vs test data

An overfitted network will have adjusted better to the training data than to the unseen test data. Since W_e captures overfitting, it should be possible to tell apart training and test data given measurement outputs. In the plots, we compare the distribution of our measure on 250 test (blue) and 250 training (purple) points from the same dataset. On each point, we sample 500 noise points and compute W_e with low σ to capture small differences. The results are plotted in Figure 5.2.

Results. In Figure 5.2a, training and test data differ for $\sigma \leq 0.025$. More specifically, the training data (purple) shows less spread than the test data. This translates to less entropy around training data, or more stable classification, as expected. To illustrate that the results are not cherry picked, we show the results for three different random draws on CIFAR in Figure 5.2b. At $\sigma = 0.0025$, the results become very obvious and the difference of the spread is clearly visible. In contrast to Fashion MNIST, however, differences in the measure at smaller σ can be spotted, but are less pronounced.

Implications. The question whether or not a data point formed part of the training data is critical knowledge for many applications such as medical sciences, social networks, public databases, or image data (e.g., facial recognition). The corresponding attack is called membership inference [111, 118, 145]. However, so far, even attacks with weak attackers [111] rely on output probabilities, and a straight-forward defense is to only output the one-hot labels. Our technique allows to circumvent these defenses, as the W_e can approximate a distribution that contains membership information based only on the one-hot output.

Measuring Overfitting during training

As overfitting unfolds during training, we study how W_e develops during training. We connect this with a study of different training variations that either increase or decrease overfitting.

Setting. We train 15 networks on the Fashion MNIST dataset. In each iteration, we sample $2,000$ noised points. As we expect the surface to vary drastically during training, we choose a large $\sigma = 0.15$ around a given test point and compute the measure over the whole test set for stability. We choose five settings, where the first two cases are used to investigate the effect of the number of sampled points when computing the measure. We also investigate adversarial training, where one adds adversarial examples with the correct labels during training. This training makes the network more robust, and has been shown to reduce overfitting [86]. Furthermore, we examine a backdoor scenario. Here, the attacker introduced a pattern in the training data which is always classified as the same class [77, 148]. The last setting is adversarial initialization [74]. Such an initialization is generated by fitting random labels before training on the correct labels. The authors showed that adversarial initialization increases overfitting. More concretely, even though the training error of such networks is low, the test error tends to be high. We do not compute the W_e during the pre-training with random labels.

Plots. Figure 5.3 depicts all five scenarios. The lines show the mean, the error bars the variance over the different runs. The blue line ('2000 samples') denotes the baseline networks, light blue ('5000 samples') the same baselines, where we sampled 5,000 noised points. The adversarial training is visualized as a red, backdoor as a yellow, and adversarial initialization setting as a green line.

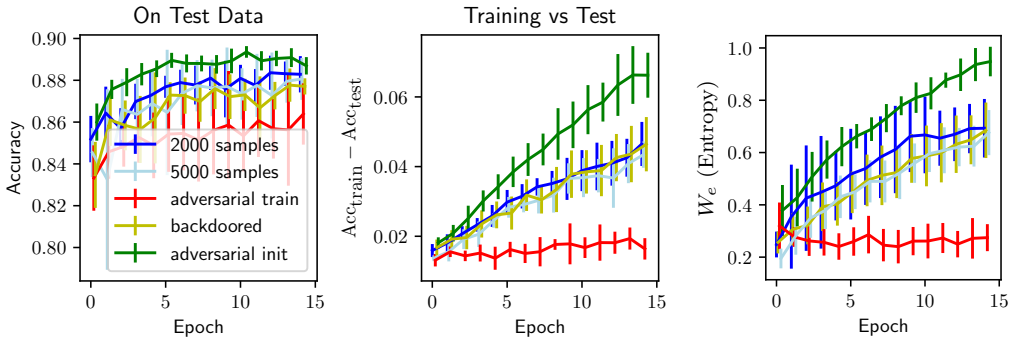
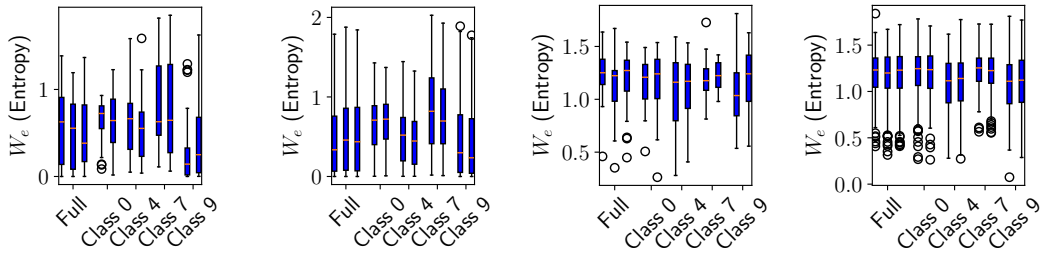


Figure 5.3: W_e on a several networks during training on Fashion MNIST. σ is set to 0.15.



(a) F-MNIST, 25 points. **(b)** F-MNIST, 250 points. **(c)** CIFAR10, 25 points. **(d)** CIFAR10, 250 points.

Figure 5.4: Influence of number of points and classes on W_e. We show three (full data)/two (classes) different random draws for each setting to show variability.

Results. The left plot shows clean test accuracy. There is no a specific point at which the test error decreases. Hence, we also depict the difference between the training and test accuracy (middle plot). The adversarial training case (red line) is the lowest, corresponding to the least overfit networks. On the other hand, the adversarially initialized networks are the highest. The normal and backdoored models appear in the middle. This corresponds to our expectations about adversarial training and adversarial initialization. This order is preserved for W_e in the right plot.

Class-wise differences

One might be tempted to think that a measure for overfitting should be class-independent. Yet, as accuracy for different classes may differ, so may overfitting properties. We study W_e on individual classes, connecting it with experiments to test stability depending on the number of test points used to compute W_e. In the previous experiments, measurements were done on 250 points or the whole test data. We now compare different draws of a very low size, 25, and the initial size 250 to confirm that the latter is sufficiently stable. To reduce the amount of plotted data, we randomly chose 4 classes for the plots in Figure 5.4.

Results. On Fashion MNIST, W_e varies when using 25 points to compute the measure (Figure 5.4a). For example, the mean of the measurement changes more than 0.4 when test points from all classes are used. An extreme case is class 9 (ankle boot), where one measurement has half the spread of the other measurement. The measure is already fairly stable around 250 test points (Figure 5.4b). Here, the largest change of mean is around 0.05. However, stability could be improved further if necessary by using more test points. On CIFAR, the results are analogous as visible in Figure 5.4c and Figure 5.4d.

Conclusion of empirical study

We showed that W_e indeed captures overfitting in different settings. After training, both training and test data exhibit differences in the measure. W_e also reflects training, and in particular configurations during training that affect overfitting like adversarial training or adversarial initialization.

Resulting complexity. 250 points without labels are sufficient, where we sample $n = 250$ points around each. Hence, the time complexity of computing the measure is $250 \times n$ or $O(n)$. The sample complexity is even lower with only 250 *unlabeled* points.

Overfitting and backdoors

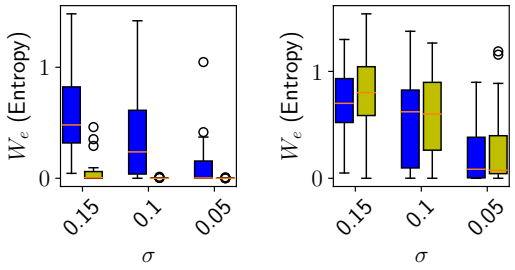
To further show the practical usefulness of our measure, we investigate a common hypothesis from adversarial machine learning. This hypothesis by Wang et al. [135] states that backdoors are enabled by overfitting. A backdoor is a particular pattern hidden in the training data, which can later be used to evade the trained classifier at test time [77, 148]. Examples for such patterns on visual recognition or classification tasks are given in Figure 5.5. Most of these particular backdoors were also used by Chen et al. [18] and Gu et al. [46]. In this section, we first present that W_e allows to distinguish data points with and without a functional trigger. We then investigate under which circumstances detection given an functional trigger is possible.

Figure 5.5: Possible backdoors on Fashion MNIST.

Attacker and hypothesis. In some cases, the attacker can only control the victim's data. As the victim might inspect this data later, the amount of injected poisons is traditionally very small (roughly 250 for >50,000 training points) [18, 132]. In other cases, the attacker trains the model, and the trained model is then handed over to the victim. In this setting, the victim is not able to inspect the training data, and, thus, the attacker can poison a larger fraction of the training data to achieve better results (10-20% in for example [18, 135]. We investigate both settings and use 250 points for the former case and fix the percentage to 15% for the latter case in our experiments. Our trigger is in a fixed position, and is added to all classes c_i except the target class c_t, hence $c_i \neq c_t$. Previous work [135] has indicated

that backdoors are related to overfitting, equating the backdoor pattern to a *shortcut* between the target and victim classes. In this setting, the model has been poisoned such that *any* input plus the trigger will result in the desired target class.

Experimental setting. We train a slightly larger network on Fashion MNIST to allow the network to fit the backdoors well: a convolution layer (64 3×3 filters), a max-pooling layer (2×2), another convolution layer (32 3×3 filters), two dense layers with 500 neurons are followed by a softmax layer with 10 neurons. The networks perform with around 90% accuracy on benign data, and with 99% on backdoors. On the CIFAR10 data, we use a smaller ResNet18, on which performance we will comment below after the initial experiments.



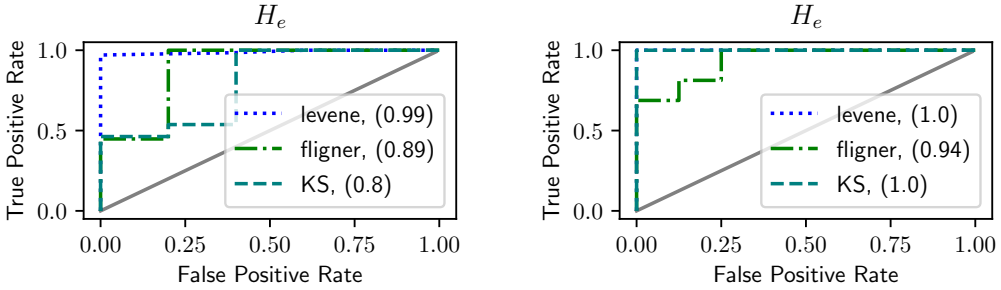
(a) Functional trigger **(b)** Benign data

Figure 5.6: W_e and backdoors on Fashion MNIST. We compare clean test data (blue) and a functional trigger (99% accuracy)/an unused trigger (9% accuracy on poisoning labels, 89% on clean labels).

$\mathbf{W_e}$ and backdoors. If backdoors indeed rely on overfitting, we expect W_e to be high around backdoors, as the backdoors are close to the decision boundary. We thus pick as few as 25 test points and compute W_e once on these clean points (blue). We are using the same 25 points with the leftmost backdoor pattern from Figure 5.5 added (yellow). As we are investigating the *difference* in behavior for those 25 points, such a low number is indeed sufficient. As a sanity check, we also evaluate an unseen/random pattern (right plot in Figure 5.6), the fourth trigger from Figure 5.5. In contrast to what the high accuracy might suggest, the model does not overfit but rather underfits backdoors: W_e is consistently low for the functional trigger, in particular at large σ. The classification output remains consistent regardless of the noise added to compute W_e. Hence, the backdoors are not close to the decision boundary, and are therefore underfit. This behavior is the attacker's goal: as soon as the backdoor is present, other features become irrelevant.

To prevent in the following evaluation that the overfit MNIST models influence the performance of W_e, we train the CIFAR networks few epochs to underfit the benign data at 63% accuracy. Still, the CIFAR networks exhibit a very high accuracy, $> 99\%$, on backdoors.

Detection mechanism. The differences in W_e above are quite pronounced and should be detectable. Statistical tests with the H_0 hypothesis that both populations have equal variance are the Levene [93] and the Fligner test [33]. The latter is non-parametric, the former assumes a normal distribution but is robust if the actual distribution deviates. Alternatively, the Kolmogorov-Smirnov (KS) test can be used with the mean-based test statistic proposed by [53]. The KS test evaluates the H_0 hypothesis that both samples are from the same distribution. To evaluate the performance of the tests with



(a) CIFAR10, 15% poisons **(b)** Fashion MNIST, 250 poisons

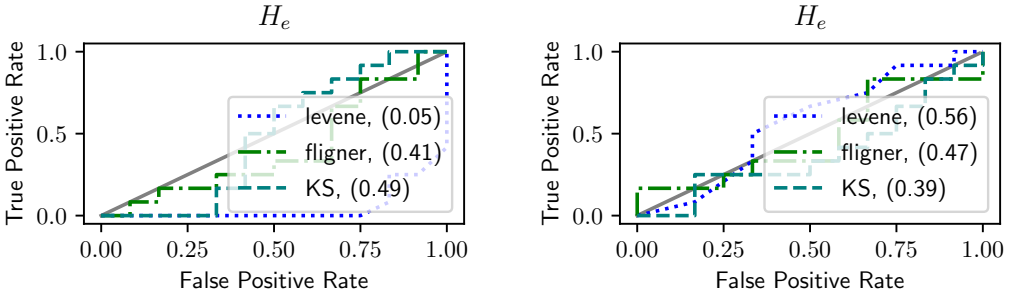
Figure 5.7: Performance of statistical tests to detect backdoors using W_e.

W_e, we train three networks on clean data and nine networks with different implanted backdoors. Of the latter backdoored networks, three are trained with backdoors one, three and four each from Figure 5.5. To obtain a valid false positive rate, we test for all five backdoors of Figure 5.5 on all networks. We then compute a ROC curve and the AUC value given the p-values and the ground truth of each test. In some cases, the test performs better when we remove outliers (defined as above or in [37]). In case that all points have the same value, we do not remove any point.

Performance of detection. Our results are depicted in Figure 5.7. In general, the performance of the tests is very good. The Levene test consistently performs best. The worst AUC (0.8) occurs using the CIFAR10 dataset, where 15% of the data is poisoned. The performance of the Fligner (0.89) and Levene test (0.99) is higher. On Fashion MNIST, the worst AUC is 0.94, although only 250 points in training are poisoned. The other two tests show perfect performance at AUC 1.0.

False negatives? To validate the above results, we repeat the previous experiments with universal examples, instead of backdoors, added to the test data. Universal adversarial examples are perturbations that are crafted after training and lead misclassification when added to several of data points [87]. They are thus similar in that they lead to missclassification, yet differ in that the network has not been trained on these patterns. Ideally, W_e should not confirm universal perturbations as backdoors: universal perturbations are unknown to the network, as they were computed after training. We depict the results in Figure 5.8. Here, a true positive corresponds a universal perturbation classified as backdoor. Hence, we expect the performance to be low: The perturbation is not a backdoor, and should not be confirmed. The tests are now close to a random guess, with the exception of the Levine test with an AUC of 0.05 on CIFAR. The other two tests are close to a random guess with AUC of 0.41 (KS) and 0.49 (Levene). On Fashion MNIST, the tests are also close to random guess, with KS (0.39) being furthest away, followed by Levene (0.456) and Fligner (0.47). We conclude that the test does not confirm universal adversarial examples as backdoors.

False negatives! We now test a method to generate backdoor candidates given a network by Wang et al. [135]. The idea behind their algorithm is to generate a



(a) CIFAR, 15% poisons **(b)** Fashion MNIST, 250 poisons

Figure 5.8: Performance of statistical tests to wrongly detect universal perturbations as backdoors. In contrast to previous plots, lower is better.

perturbation with a minimal mask that, when applied to any image (in a given input set), will cause *all* images to return the same fixed class. In the original work, the size of the perturbation indicates if the trigger is functional or not. We generate a trigger candidate for each class on clean and backdoored models. As the generated points are in fact universal perturbations with the additional constraint of a small perturbation and consistent missclassification, we expect our measure not to confirm these samples: the network has not seen them during training. As before, we add the candidates to the 25 test points, compute the W_e, and feed the measure into the test. In contrast to our expectations, however, the test detects all of the inputs with the crafted perturbation as backdoors with the same, very low p-value. As a sanity check, we add the patterns found to the training data (which has not been used to generate them) and find that the accuracy on the targeted class is 100% in all cases. We conclude that our measure wrongly confirms a perturbation crafted by the method of Wang et al. [135] as backdoors, although the generated patterns have not been maliciously added to the training set.

Conclusion. Our measure, when combined with a test, reliably detects implanted backdoors. We further confirm that perturbations computed at test time (universal adversarial examples) are not detected. Yet, crafted backdoor candidates are found that manage to fool our detection. Hence, even benign networks contain variants of backdoors that emerge during training.

Conclusion

In this chapter, we have introduced the W measure that quantifies output sensitivity and overfitting. Intuitively, such a measure can be used to determine membership, as training data will be fitted better than unseen test data. Furthermore, we show that W_e is able to detect backdoors. Since these backdoors are reliably (mis)classified, they are underfit, which is captured by the measure. However, our experiments show as well that there are naturally occurring underfit patterns in benign neural networks, too.

The latter finding has strong implications for all mitigations that rely on either generating a trigger, for example the work by Wang et al. [135] or Xiang et al. [138] and

mitigations that rely on the a stable classification output for backdoors, like for example the work by Liu et al. [78]. Since backdoors are not the only patterns exhibiting such reliable (mis)classification under perturbations, there is no guarantee that the confirmed pattern is indeed a bakdoors and was inserted by an attacker during training.

6
Adversarial initialization

Introduction

In the previous chapters, we dealt with intrinsic properties of models and attacks that made achieving security difficult. We now add a new layer, thereby showing how security problems can also emerge when taking into account the broader application of an algorithm. As an example, we take advantage of the complexity of today's libraries, and alter the initialization function of a neural network. We show that this new training attack has devastating effects [P4]. We name this threat adversarial initialization, and although such an initialization is in theory easy to recognize, it requires that the victim *is aware* of the corresponding threat. As we show in a study, otherwise failure modes of adversarial initialization are generally attributed to other causes like bugs or issues with the data.

Adversarial initialization

We introduce attacks that alter the initial weights of a neural network. The goal of the attacker is to decrease accuracy drastically, or to increase training time. Ideally, this is done in a stealthy way: the victim should not spot the attack.

Before we discuss specifics and the generalization of our attacks, we motivate our approach by discussing its most basic version. The following equation represents two consecutive layers in a fully connected feed-forward network with weight matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{\ell \times m}$, corresponding biases $\mathbf{a} \in \mathbb{R}^m$ as well as $\mathbf{b} \in \mathbb{R}^\ell$, and ReLU activation functions

$$\mathbf{y} = \mathrm{ReLU}\left(\mathbf{B}\,\mathrm{ReLU}(\mathbf{A}\mathbf{x} + \mathbf{a}) + \mathbf{b}\right) \quad . \tag{6.1}$$

This vulnerable structure or similar vulnerable structures (like two consecutive convolutional layers) can be found in a plethora of typical DNN architectures. We assume that the neurons are represented as column vectors. The formulation for a row vector is completely analogous. We further assume that the components of $\mathbf{x}$ are positive. This corresponds to the standard normalization of the input data between 0 and 1. For input vectors $\mathbf{x}$ resulting from the application of previous layers it is often reasonable to expect an approximately normal distribution with the same characteristics for all components of $\mathbf{x}$. This assumption is (particularly) valid for wide previous layers with randomly distributed weights because the sum of many independent random variables is an approximately normally distributed random variable due to the central limit theorem [100].

The idea behind our approach is to make many components of $\mathbf{y}$ vanish with high probability and is best illustrated by means of the sketches in equation 6.2 and equation 6.3. The components of the matrices and vectors are depicted as little squares. Darker colors mean larger values. In addition, hatched squares indicate components with a high probability of being zero.

In matrix $\mathbf{A}$, the largest components of the original matrix are distributed in the lower $(1-r_A)m$ rows. $r_A \in \{\frac{1}{m}, \frac{2}{m}, ..., 1\}$ controls the fraction of rows that are filled with the "small" values. The small and often negative components are randomly distributed in the upper $r_A m$ rows. The products of these negative rows with the positive $\mathbf{x}$ are

likely negative. If the bias **a** is not too large, the resulting vector has many zeros in the upper rows due to the ReLU-cutoff.

$$\text{ReLU}\left(\underbrace{\begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \end{bmatrix}}_{\text{matrix } \mathbf{A}} \underbrace{\begin{bmatrix} \\ \\ \\ \end{bmatrix}}_{\mathbf{x}} + \mathbf{a}\right) = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix} \tag{6.2}$$

Next, a similar approach can be used with matrix **B** to eliminate the remaining positive components. Let r_B control the fraction of "small" columns of **B**.

$$\text{ReLU}\left(\underbrace{\begin{bmatrix} & & & & \\ & & & & \\ & & & & \\ & & & & \end{bmatrix}}_{\text{matrix } \mathbf{B}} \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} + \mathbf{b}\right) = \underbrace{\begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}}_{\mathbf{y}} \tag{6.3}$$

In summary, we concentrate the positive contributions in a few places and "cross" **A** and **B** in order to annihilate them. For the typical case of weights drawn from a zero mean distribution, $r_A = r_B = \frac{1}{2}$ effectively kills all the neurons and makes training impossible.

The probability for obtaining a matrix like **A** in equation 6.2 by chance is very small and given by ${}^{((r_A mn)!((1-r_A)mn)!)}/_{(mn)!}$.

With the general idea of our attack in mind, we can now discuss specifics. A complete blockade of the entire network obviously contradicts the idea of stealthiness because at least some learning is expected by the user. The prototypical attack must thus be "weakened" in a controlled manner to be stealthy.

Soft knockout attack. The first way of controlling the network capacity is by varying r_A and r_B in such a way that some but not all of the neurons have some non-vanishing probability of being non-zero. This is achieved by choosing $r_A < 1/2$ or $r_B < 1/2$, respectively $r_A \gg 1/2$ or $r_B \gg 1/2$.

Shift Attack. As an alternative, we can choose $r_A = r_B = 1/2$ and shift the columns of **B** periodically by s positions. In a fully connected network, this corresponds to s active neurons, yielding specific control over capacity.

Algorithmic procedure. We formalize both approaches in Algorithm 6.1 for fully connected layers. In both cases, the attacker only alters one weight matrix at a time, in the order one would initialize the weights in the model. To perturb a weight matrix, we first obtain the $(100 \times r)\%$ smallest weights, denoted as **S** (line 4). We denote the remaining, larger weights as **L** (line 5). For the shift attack, we instead split the weight matrix in negative (line 7) and non-negative (line 8) components. Depending on the variable *cross* which is flipped each round (line 18), we reorder the weights. For the first and all unevenly indexed matrices, we align the small weights in the upper rows and then fill up with the larger weights (line 13). For all evenly indexed matrices, we cross

the components: The first columns contain the large weights, and we fill up with the small weights (line 11). When using the shift attack, In all uneven cases, we additionally shift the rows by s positions to activate s neurons (line 15). In this formalization, we skip the details of the exact reshaping operations needed to obtain matrices of the correct shape. We also leave aside that columns and rows might be filled partially with small and large weights.

Algorithm 6.1 Soft Knockout/Shift Attack. Given a stream of weights $\mathcal{W} = \{\mathbf{W}_1, ...\}$, parameter $r \in [0, 1]$, and $s \in \mathbb{N}$, outputs permuted, harmful weights impeding training.

Require: $\mathcal{W}$, r, s
1: cross $\leftarrow$ False
2: **for** $\mathbf{W}_i \in \mathcal{W}$ **do**
3: **if** soft knockout attack **then**
4: $\mathbf{S} \leftarrow$ smallest $r|\mathbf{W}_i|$ components of $\mathbf{W}_i$
5: $\mathbf{L} \leftarrow$ largest $(1-r)|\mathbf{W}_i|$ components of $\mathbf{W}_i$
6: **else** {*prepare shift attack*}
7: $\mathbf{S} \leftarrow$ negative components of $\mathbf{W}_i$
8: $\mathbf{L} \leftarrow$ non-negative components of $\mathbf{W}_i$
9: **end if**
10: **if** cross **then**
11: $\mathbf{W}_i \leftarrow \begin{pmatrix} \mathbf{L} & \mathbf{S} \end{pmatrix}$
12: **else**
13: $\mathbf{W}_i \leftarrow \begin{pmatrix} \mathbf{S} \\ \mathbf{L} \end{pmatrix}$
14: **if** shift attack **then**
15: shift rows of $\mathbf{W}_i$ by s positions periodically
16: **end if**
17: **end if**
18: cross $\leftarrow \neg$ cross
19: **end for**

Adversarial initialization for convolutions. Particular care has to be taken when attacking convolutional layers. Yet, the idea of weight permutation and matrix crossing works in a very similar way. We formalize the attacks for convolutional weights represented as 4-dimensional tensors: filter height $\times$ filter width $\times$ number channels $\times$ number filters. This requires a different sorting of the components than for fully connected layers. The procedure is illustrated for two consecutive convolutional layers with a one-channel 4×4 input and a four-channel 4×4 output The smallest weights are randomly distributed over the first half of the *filters*, resulting in a very likely deactivation of half the channels. For each filter of the second layer, half the *channels* are equipped with the small weights, so that the negative filter channels are applied to the positive input channels. The positive filter weights are applied to the deactivated neurons, and do not contribute to the sum over all channels for each filter. Thus, deactivation of all output channels is probable.

Given this layout, we shift the channels of a filter of the second layer in order not to block the whole network. Compared to the previously discussed shifting attack, we have more degrees of freedom: a shift per filter and the number of filters where to apply shifting. The same holds for the soft knockout attack, where we specify on how many filters in the even layers the permutation is applied.

Empirical evaluation

We evaluate the previously derived attacks. We first detail the setting, datasets and architectures and explain how we illustrate findings.

Setting. We evaluate two different kinds of architectures, fully connected networks and convolutional networks. All our fully connected networks contain $n/2$ neurons in the first hidden layer, where n is the number of features. The second hidden layer has 49 neurons for the two MNIST tasks. As an example for a convolutional architecture, we use LeNet on CIFAR. All networks are initialized with He initializer [49] and constant bias. The fully connected networks are trained for 300 epochs on both MNIST variants. LeNet is trained for 200 epochs. We optimize the nets with the Adam optimizer with a learning rate of 0.001.

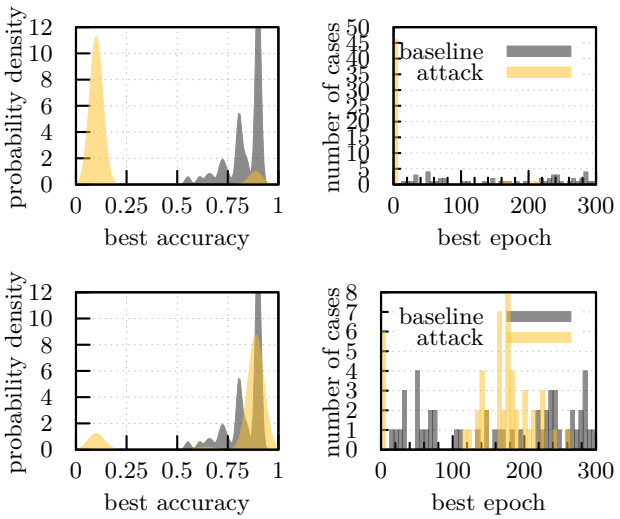


Figure 6.1: The soft knockout attack allows little control over final accuracy: Fashion-MNIST, fully connected network, $r = 0.25$ (upper) versus $r = 0.2$ (lower).

Presentation of results. We are interested in how our attacks affect the probability to get a well performing network after training. Towards this end, we mainly consider two quantities: the best accuracy that is reached during training and the epoch in which it has been reached. We approximate both distributions by evaluating a sample of 50 networks with different seeds for the random initializer.[1] We plot the smoothed probability density function over the best test accuracies during training and the epochs at which this accuracy was observed. While we use Gaussian kernel density estimation for the former, the latter is depicted using histograms. Both distributions are compared to a baseline derived from a sample of 50 clean networks with the same 50 random seeds.

Knockout attack. In this attack, we control the size of the split between small and large values of the weight matrices in order not to knock out all the neurons at once.

[1] We keep the same 50 seeds in all experiments for comparability. However, due to effects from parallelization on GPUs, the accuracy might differ by up to 2% for seemingly identical setups.

The experiments show that this gives little control over performance: On fully connected networks, when $r > 0.3$ training fails entirely. When $r \leq 0.2$ the network achieves normal accuracy (however needs more of epochs). As soon as the networks have some non-vanishing chance of updating the weights (which is the idea of a soft knockout), they can recover from the bad initialization.

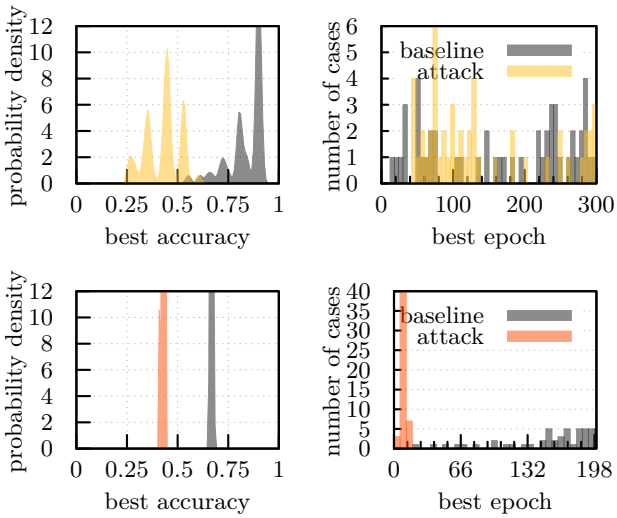


Figure 6.2: The shift attack on Fashion MNIST (upper) and CIFAR10 (lower). In both cases, shift is set to eight, for the convolutional network on CIFAR, we apply the shift to one filter.

We plot the results on Fashion-MNIST for $r = 0.2$ and $r = 0.25$ in Fig. 6.1. A parameter $r > 0.3$ leads to complete failure to learn: all accuracies are equivalent to guessing. Networks that perform with random guess accuracy usually perform best in their first iteration, and do not improve during training. This is visible as well for $r = 0.25$.We picked Fashion-MNIST to illustrate this, although it occurs in general. For slightly lower $r = 0.2$, however, most seeds achieve baseline accuracy, where training time increases on average.

Shift attack. This attack gives more fine-grained control over the network. In fully connected networks, the shift parameter is equivalent to the number of active neurons. Our experiments show that a number of 10 (MNIST)/ 12 (Fashion MNIST) neurons suffices to learn the task with unchanged accuracy. We set the shift of 4 and 8 on MNIST and Fashion MNIST (see Figure 6.2). In both cases, the maximal accuracy is around 50%, but the network still learns. On Fashion-MNIST, training time increases by around 50 epochs. This is less clear for MNIST, where several networks are failing, and achieve their best (random guess) accuracy in epoch one.

The results of convolutional networks on CIFAR10 are in Figure 6.2. We apply a shift of eight and apply it to one or sixteen filters. As for the fully connected networks, accuracy decreases strongly. The average accuracy is around 43% if one filter is affected and around 50% if the number of filters is increased to sixteen. Intriguingly, training time decreases for one filter and slightly increases if 16 filters are targeted.

Why would I care?

One might wonder how an attacker might even be able to alter the code of the library. In both security [5, 67] and ML [75, 141], trust in libraries has been recognized as a threat. A simple drive-by download is enough to infect a machine with the malicious code [68], if no corresponding defense is in place [55].

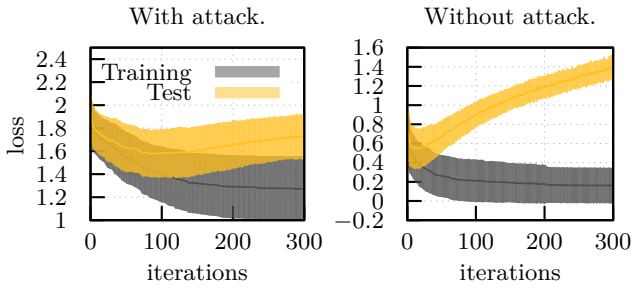


Figure 6.3: Loss during training on Fashion MNIST (fully connected network, shift is 4). Along with the achievable accuracy, the scale of the loss is unknown to the victim.

Furthermore, one might ask whether a user would actually fall for such an easy-to-fix attack as maliciously permuted weights. We argue that this hinges on the user's awareness of the attack and that current debugging routines hardly take initialization into account. In order to underpin this statement, we search for "neural networks low accuracy", "neural networks bad performance", "neural networks bad accuracy", and "neural net fail" on two popular Q& A sites for programming issues, `stackexchange.com` and `stackoverflow.com`. Due to fine nuances in meaning, we do not automate the analysis of the 332 posts. Further due to privacy concerns, we remove all user names from the stored posts and do not carry out any analysis related to users. We do not count questions without replies (22), duplicates (3), and unrelated questions (185). We consider a question unrelated if it is

1. a high level question, e.g. *what performs better, neural networks or ensembles*, or *how to deal with missing data*,
2. an implementation question, e.g. *In Tensorflow estimator class, what does it mean to train one step?* or *How to use smac for hyper-parameter selection*,
3. very specific to an application, e.g. *discussing how to improve the contours of a FedEx logo detector* or *which algorithm to use to block/unblock a gate for vehicles*,
4. a question about a specific error message, e.g. *Assertion 'cur_target >= 0 && cur_target < n_classes' failed. [...] Any ideas?* or *ValueError: Tensor Tensor("dense_2/Softmax:0", shape =(?, 2) , dtype=float32) is not an element of this graph. [...] Any ideas on why this causes this error?*.
5. a question that is of philosophical nature or entirely unrelated to machine learning.

On the remaining relevant questions, we distinguish the **overfit**ting scenario (high train accuracy, low test accuracy, 21 questions), and a **bad** performance category (both are low, 115). As some posts are ambiguous (just reporting "bad accuracy", 23), we list these separately as **unclear**.

In each of the above groups, we collect topics mentioned in the posts and categorize them to give a better overview. We only count one suggestion per category, e.g. if a user writes "use more data and split the data in a random fashion", this counts once in the data category. We then compute the percentage, e.g. for how many percent of the questions this has been suggested. Hence, 100% in the data category implies that for each question, people made a suggestion concerning data. We summarize our results in Table 6.1.

Table 6.1: Suggestions to fix bad accuracy. Percentage denotes how often reply was given in which setting.

Suggestion	Overfit	Bad	Unclear
More data, sanity check data, data split, re-weight data	71.4%	31.3%	61%
Train longer	∅	4.3%	13%
Change optimizer, loss, batch size, momentum	14.3%	27.8%	17.4%
Larger/smaller model, different type of layers, change initializer	33.3%	30.4%	26.1%
Use regularizer, cross validation, batch normalization, etc.	43%	14.8%	17.4%
Use other classifier than DNN	4.8%	7%	13%
Bugs in logic or implementation	19%	32.2%	26.1%
Number of questions evaluated	21	115	23

As expected, for the category overfitting, most replies indicate to use more data (>70%). Secondly, suggestions concerning the model size prevail. In the unclear category, most posts concern the data as well, with a tie between changes in the optimizer, momentum etc. and suggestions to fix bugs. In the bad case, the largest fraction fall as well into the data category, followed tightly by suggestions concerning the architecture of the model. This also entails changing the initializer—yet we hope to have convinced the reader in particular in this appendix that neither changing the architecture, nor changing the activation, optimizer or initializer will alleviate the attack.

In total, we find four posts and two comments which could potentially lead to the discovery of our adversarial initialization. To indicate the relevance for the community, we also write the rating on the Q&A for each question:

1. `Check your loss function, weight initialization, and gradient checking to make sure the back-propagation works in an appropriate manner` (Scenario overfitting, rating: 1)

2. `your error gradient doesn't reach initial layers! (you can check this by plotting histograms of weights in tensorboard)` (Scenario bad performance, rating 0)

3. `Look at individual layers. [...] look for layers which have suspiciously skewed activations (either all 0, or all nonzero)` (Scenario bad performance, rating 167)

4. `Did you check if the parameters get updated after optimizer.step()?.` (Scenario bad performance, rating 0)

Although all or many of these posts potentially give away the attack, the setting described is generally not the one that is caused by our attack. One concerns a setting where over-fitting takes place. In our attack, training and test accuracy generally do not diverge as the resulting small model-size prevents memorization. This is visualized in Figure 6.3. The other three posts reply to cases where the models do not learn at all, a scenario that our attacker tries to prevent to remain stealthy.

Two replies, however, are posted in the correct setting:

1. `This isn't a very good answer (thus why it's a comment) but back when I was studying neural nets you need to double, triple and quadruple check that all your variables and functions are doing and storing what you intend. One single misplaced calculation will mess up the entire system resulting poor results and ripped out hair follicles. Good luck` (Scenario Unclear, rating 0)

2. `Gradient check your implementation with a small batch of data and be aware of the pitfalls.`(Scenario bad performance, rating 1)

Concerning the first reply, it remains an open question what a user would check in detail given these broad instructions. In the second case, the user points to gradient checking. This method assumes that the gradient implementation is having a bug—in our setting, however, the bad gradients are a consequence of the small weights, and mathematically correct. Yet, inspection might give away the attack. Both posts are still far from a direct hint—we conclude that there is a lack of awareness on the importance of the initial weights.

Conclusion

In this chapter, we have introduced adversarial initialization: a training-time attack that takes advantage of the complexity of today's deep learning libraries. Although adversarial initialization is in theory straight-forward to recognize, it requires that the victim *is aware* of the corresponding threat: otherwise the effects caused by adversarial initialization are overlooked and attributed to other causes. This part illustrates the diversity threats on ML can take—the surrounding of the applied algorithm can be as important as the attack surface of the algorithm itself. More work is needed to understand the full threat surface that ML algorithms expose next to typical attacks like adversarial examples or poisoning. In particular, users need to be educated on how to recognize and address any of these threats properly.

7
Multiple attacks

Introduction

In the previous chapter, we have investigated the complexity of libraries in the context of security. The attacks presented were straight-forward to defeat when one was aware of the threat. However, recapping Chapter 4 about evasion and Chapter 5 about backdoors, both attacks are in a current arms race [3, 16, 73, 117, 128, 131].

Any broken mitigation itself points at the hardness of defending a classifier against an individual attack. In this chapter, based on [P5], we take a different approach, and study the vulnerability towards several different test time attacks at once. More specifically, we investigate a classifier that allows configuration of the decision function curvature, and show that for individual test time attacks, a seemingly secure configuration can be found. This configuration, however, will be vulnerable towards a different attack.

We start the chapter with a formal analysis showing the vulnerability of the classifier studied in this chapter—Gaussian processes (GP)—towards evasion. Furthermore, we show how attacks concerning intellectual property (IP) like model stealing or membership inference are related on a Gaussian process. To ease formal analysis, we use GP regression for the formal analysis. We then carry out the empirical study linking decision surface curvature and vulnerability of GP classification towards different attacks.

Formal analysis of vulnerability

We take advantage that a GP allows a formal analysis. First, we show that learning or generalization enables evasion vulnerability on GP. We then study the interplay of model reverse engineering, membership inference, and model stealing.

Evasion attacks

We first define a classifier that cannot be fooled by an adversarial example. In the following, we show that a classifier fulfilling this definition, and hence a static security guarantee, is opposed to learning. We briefly define *rejection* of a classifier. A classifier can *reject* a sample, in the sense that it does not assign the given sample to any predefined class.

To define a secure classifier, we chose a covariance with compact support [106]: as the distance from the training data increases, it reaches 0. Furthermore, there is a ρ such that for all training points, iff point x' is in the closed ball $B(x, \rho)$ around a training point $x \in X_{tr}$ with radius ρ, then x' cannot be an example of another class than x. In other words, all points in the ρ-ball around training point x_i are of the same class. We formalize the secure classifier

$$f(x') = \begin{cases} y_i & \text{iff } x' \in B(x_i, \rho) \\ \text{reject} & \text{otherwise} \end{cases}$$

that cannot be fooled: Changing a sample enough to be classified as a different class means to alter x' so much that $x' \in B(x_j, \rho)$ where $y_i \neq y_j$. Then, by our definition,

x' is a valid instance of this other class and not an adversarial example. This secure classifier is equivalent to a GP given the following conditions: *First*, GP has a rejection option based on ρ. *Second*, writing $k(x_i, x_j)$ for the covariance between x_i and x_j, there is no point x' such that for two distinct training points x_i,$x_j \in X_{tr}$ both $k(x_i, x') > 0$ and $k(x_j, x') > 0$.

In other words, we require that GP is able to reject a sample. This can be achieved by setting a threshold on GP's similarity. Condition two states that the similarity between any two training points is zero, independent of their class. Such a GP, however, has as covariance matrix the identity matrix, as the similarity between any two points is zero. Such a covariance matrix does not allow any learning [85]. The details of this equivalence can be found in a long version of [P5]. Assuming that the second condition does not hold, training points jointly influence classification and the GP regression (GPR) generalizes.

Theorem 1. *Either GP's covariance K is similar to the identity matrix I, or $K \neq I$ and learning occurs. Then, GPR potentially classifies areas outside the ρ-balls. Hence, for a test point x' and its corresponding output p, $p > \rho$ or $p < -\rho$ although $k(x_i, x') < \rho$, where x_i is the closest training point.*

Proof. To be classified, we need a classification output $p > \rho$ or $p < -\rho$. We start with the first case, and write

$$p \leq \sum_i (\rho - \kappa_i) * [K^{-1}]_i * 1 \ , \tag{7.2}$$

where $[K^{-1}]_i$ is the sum over the inverted covariance matrix column corresponding to point i. Before inversion, this column contains the similarities between i and all other training points. So far, we have ignored that we need a test point to obtain this prediction. Without loss of generality, we pick x' which maximizes the sum under the restriction that x' is in none of the ρ-balls: hence $\rho - \kappa_i$, the covariance to ρ-ball$_i$ is κ_i.

There are two cases. In the first, $p \geq \rho$ and we classify outside the ρ-ball. In the second, $p = 0$ or $0 < p < \rho$. As we choose the maximal x', there are no other points for which $p > \rho$. Then GPR is still secure: no area outside the ρ-ball is classified, as the output is below the defined threshold. It remains to be shown, however, that there is no contradiction for the opposite class. We proceed analogously with an x'' that is chosen to minimize the sum. ■

We used in the proof that the minimal output of a point chosen to maximize the sum is zero. Analogously, the maximal value when minimizing the sum is zero as well. This holds due to the abating property of the kernel: As we move away from the data, eventually all similarities become zero, thus the sum is zero as well. We conclude that generalization enables test time attacks such as evasion or adversarial examples.

Further test time attacks

As GPs are an instance of lazy learning, in general all training points and parameters are used during inference. Intuitively, this should ease extraction for the attacker. As we show here, this need not be the case. We briefly recap the attacker's goal in each

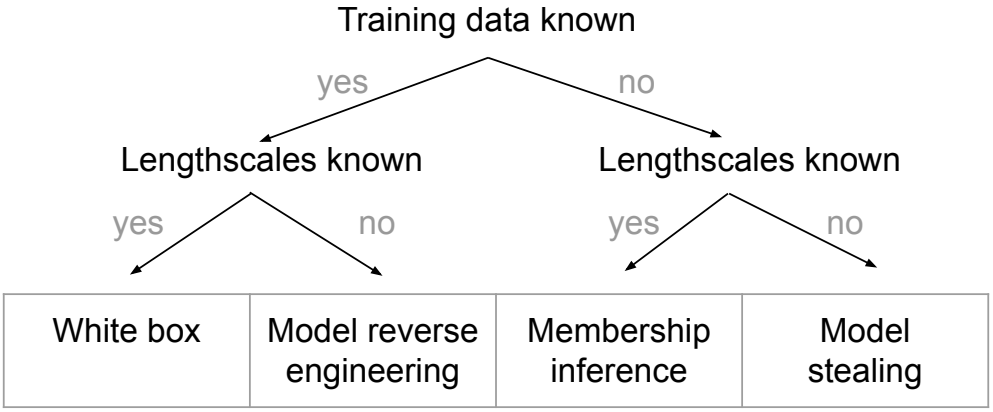


Figure 7.1: The relationship of IP based attacks on GP models.

attack. In **model reverse engineering**, she wants to obtain the lengthscale(s), in **membership inference** the full or partial training data, and in **model stealing** both lengthscale(s), and full training data. These attacks are strongly related for GPs, as visible in Figure 7.1.

We recap how classification is computed in GPR. The posterior mean y^* is given as

$$y_t^* = K_{x'}^T K_{\text{tr}}^{-1} Y_{\text{tr}} = \sum_i k(x' - X_i) * K_i^{-1} * Y_i \ , \tag{7.3}$$

where we iterate over the n training data points. The covariance metric k is parametrized using l and σ^2 when using the RBF kernel. We write K for the covariance matrix resulting from k and denote the inverse as $^{-1}$. As lazy learning is used, one might suspect that we can simply *extract* the stored parameters and training data. For example, independent from the used kernel, we unfold this sum and add the observed output vector of a GP to obtain an equation system. For simplicity, assume that $Az = y_o$, where z refers to the parameter the attacker wants to retrieve, and y_o is the output observed from the targeted GP. Further A denotes the matrix specified in equation 7.3, without z.

The interested reader will have noticed, however, that this equation system solves for unknowns in the number of training points whereas we need an equation system solving along the features dimensionality. In terms of the above equation, we are actually interested in $A^T z = y_u$, where y_u is an output per feature (where feature and data point dimension are swapped, or X^T). Hence, y_u is not an output for any GP trained on X: it corresponds to a label per feature. In the original task, the features are "lost" in the kernel Hilbert space inside the GP, and the attacker has no equation system since there is no y_u.

The existing equation system can only be used to determine the lengthscale iff there is only one global lengthscale set, and the GP has no other unknown parameter. Otherwise, the equation system is not properly specified, and no analytic computation is possible. We thus conclude that lazy-learning, albeit counter-intuitively, is not less privacy resilient that other classifiers. Instead, however, the attacker can take advantage

Attacker	F	A	I	L
Evasion	✓	X	X	X
Model Extraction l/lengthscale	X	✓	X	✓
Model Extraction k/kernel	X	✓	X	X
Membership Inference	X	✓	✓	✓
Model Stealing	X	✓	X	X

Table 7.1: Attackers knowledge according to the FAIL model. The symbol ✓ denotes 'known' or 'is altered', X the opposite.

that GPs are deterministic. A GP with the same parameters and data always yields the same output. In the following, empirical section we evaluate this type of attack.

Conclusion

GP, as it learns, is vulnerable to evasion attacks. Concerning IP-related attacks, we can exclude the possibility that the attacker analytically determines training data or lengthscales, with the exception of a single learned parameter for all dimensions (for example in a linear kernel).

Empirical study of vulnerability

We now describe our complementary empirical study. We start with the setting including data-sets, implementation, and parameters. Afterwards, we detail the results on evasion, model reverse engineering and membership inference.

Threat model

We specify the different adversaries of our empirical study. In the FAIL [126] model, F denotes the attacker's knowledge about the features. A denotes knowledge about the algorithm applied and I about the training data. L summarizes whether changes to the data by the attacker are constrained. A succinct overview for each attack is given in Table 7.1.

Evasion. Our attacker knows and changes all features, but is oblivious about the training data and the algorithm.

Model reverse engineering (l). The attacker only knows a GPC with an RBF kernel is used. The data knowledge varies from black-box to white box, without modifying samples.

Model reverse engineering (k). The second attacker only knows GPC is applied. Yet, she uses the zero and the ones vector as input, and is thus not constrained on features.

Membership inference. We assume a worst case scenario, where the attacker obtained a large fraction of data labeled as part of the training set. The attack is not tailored for the learning algorithm, and does not alter the input.

		short l			long l		
Data-set	n	l	Acc_r	Acc	l	Acc_r	Acc
Hidost	500	.5	98.4	98.4	1.9	97.7	99.6
Drebin	750	.5	54.4	94	1.9	94.8	94.8
Spam	500	.3	92.6	91.7	5	92.7	90.2
Bank	500	.3	100	100	2	100	100
MNIST91	500	1	98.9	98.3	8	99.5	99.5
MNIST38	500	1	93.4	93.4	8	97.4	97.1
SVHN91	1500	8	85.4	88.5	16	83.8	87.6
SVHN10	1500	8	88.7	88.7	16	88.7	88.7

Table 7.2: Number of samples used in training n, lengthscales l and accuracies (rejection if $y_t^* = 0$, written Acc_r).

Model stealing. In our setting, model stealing on a GP can be seen as a combination of the previous two attacks.

Experimental setting

We first describe the general setting. Specifics are given jointly with the corresponding attacks.

Implementation. We use Python and GPy for the Gaussian Process approaches [45]. We show further information on the trained GPCs in Figure 7.2, such as the number of training samples and lengthscales used and achieved accuracies. To obtain adversarial examples, we use Tensorflow [82] and the Cleverhans library 1.0.0 [44] for DNN, and other public implementations [P2, 17].

Parameter choices. We train our GPC using the RBF kernel with a predefined lengthscale. This GPC is fitted until convergence or for 100 iterations. For each task, we chose two lengthscales that achieve similar accuracy (see Table 7.2). More details on how we determined the two used lengthscales can be found in a long version of [P5].

Evasion / adversarial examples

We expect that a GP with a long lengthscale misclassifies fewer adversarial examples: A larger perturbation δ is needed to cause the same change in the output.

Setting. To obtain adversarial examples independent of the specific curvature, we do not craft on the GPCs tested. We instead transfer FGSM, JSMA and $\mathbf{L_x}$ attacks from deep neural networks, linear SVM and a GPC substitute. Our intention is to study a wide range of attacks, including optimized, unoptimized, one-step and iterative attacks as well as different metrics (L_0, L_2, and L_∞). We summarize all attacks based on the

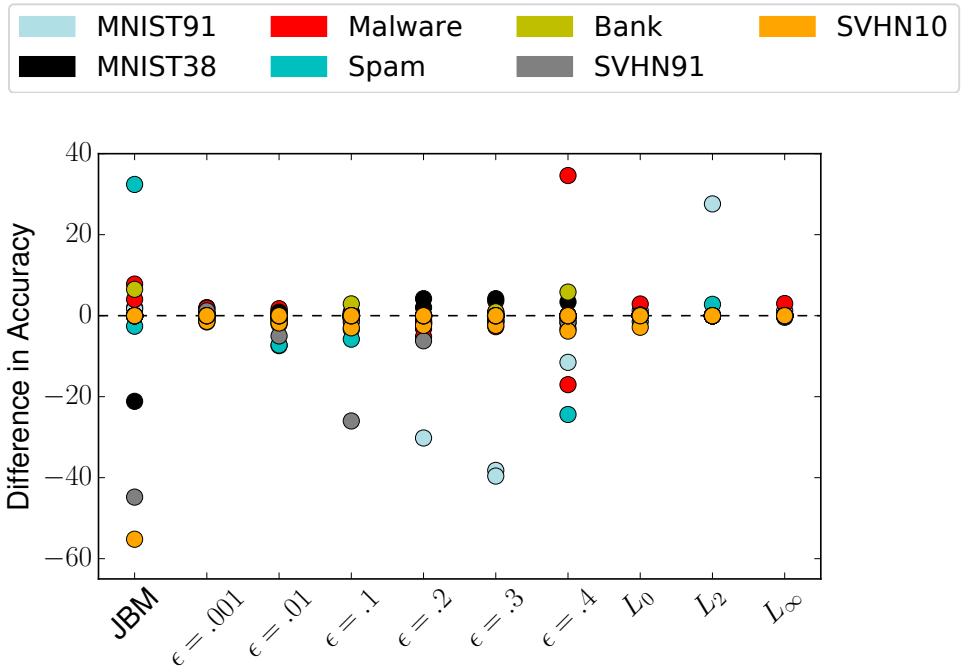


Figure 7.2: Vulnerability and curvature in GPC. Above zero denotes that more examples are correctly classified by a GPC with long l, below zero with short l.

Jacobian in JBM, sort FGSM according to ϵ and plot the $\mathbf{L_x}$ attacks according to the norm optimized (for example L_2 for the L_2-norm attack).

We compare how well the previously chosen lengthscales recover the correct class when facing adversarial examples. In our plots, a value above zero denotes that the shorter lengthscale classified more data correctly, where the numbers are difference in absolute percent. Below zero, a longer lengthscale (flat curvature) performed better.

Results. We plot the results of our experiments in Figure 7.2. A short lengthscale generally classifies more adversarial examples as their original class. In particular on L_∞ attacks (with $\epsilon > 0.01$), a short lengthscale performs better. A long lengthscale is advantageous for optimized attacks like L_2.

We also investigate how lengthscale affects rejection, as our preliminary results show only a slight advantage for steep curvature GPs without rejection. In Figure 7.3, a negative number denotes how much absolute percent the reject performs better compared to a classifier without reject. A positive number means that accuracy for rejection is worse. There is no difference in vulnerability to evasion for a long lengthscale. For a short lengthscale, the effect is positive or neutral, with only two negative cases. These two cases stem from the highly imbalanced Hidost data set. By chance, the assignment of the forced classification was in favor of the larger class.

Conclusion. Only classifiers with steep decision functions benefit from rejection. We hypothesize that a short lengthscale allows for larger areas where the rejection area is

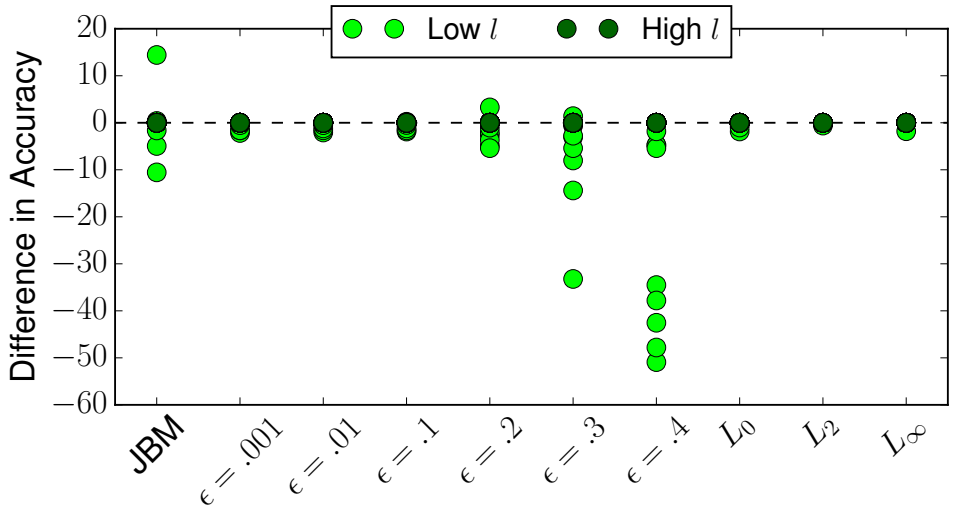


Figure 7.3: Vulnerability, lengthscale and rejection option in GPC. Above zero denotes that more examples are correctly classified or rejected by a GPC without a rejection option.

actually used, whereas a long lengthscale leads to confident classification in areas where no benign data was seen.

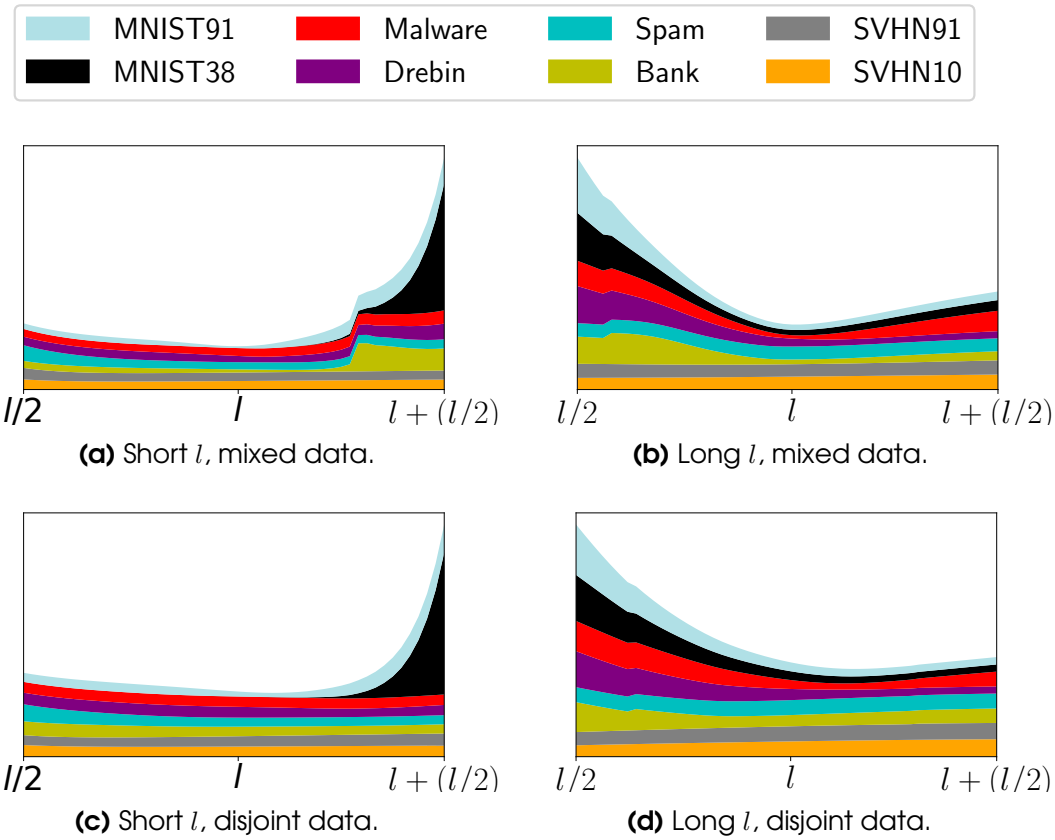
Model reverse engineering

Model reverse engineering refers to the retrieval of hyper-parameters of the model. We introduce two new attacks to reverse engineer GP's lengthscale and kernel.

Setting (lengthscale). We pick the same lengthscales as before and evaluate whether the attacker is able to determine the lengthscale of a target GP. The attack is a binary search to obtain l. The distance between the outputs of two GPs shrink as the lengthscale chosen by the attacker, l_a, approaches the original lengthscale l. We evaluate three settings: Training GPC on the same data as the victim, mixed (half/half) and disjoint data. In each setting, we train 50 GPCs, starting with a lengthscale $l_{a=0} = l/2$ and increasing the lengthscale in 50 steps of $(1/50)l$. We then compute the absolute difference between the outputs of the GPCs on hold out, unused test data.

Results (lengthscale). As GPs are deterministic, all distances decrease towards the original l when the training data is fully known. We thus omit plotting these results, and study the more interesting cases of mixed data.

For mixed data (upper plots of Figure 7.4), the results are less clear. In general, distances decrease towards l. Given a long lengthscale, the distances are smallest around l. An exception are SVHN and Spam, where the distances remain constant for all l_as. On Drebin, the distances are smallest around $l + (l/2)$. The results vary for a short lengthscale: for some data sets (MNIST91, Bank) the distance is closest to l, For others (including SVHN and Malware), the smallest distance is $l/2$.



(a) Short l, mixed data. **(b)** Long l, mixed data.

(c) Short l, disjoint data. **(d)** Long l, disjoint data.

Figure 7.4: Normalized, absolute differences in output for different data sets when binary searching a GP's lengthscale. x-axis is L_a; hence at l, lengthscales are equivalent.

In case of disjoint data sets (bottom plots of Figure 7.4), the results are even less pronounced. The distances slightly decrease towards the original lengthscale, yet the average minimum is at a lengthscale $> l$. In case of a short lengthscale, there are no differences at all. An exception are the two MNIST tasks, where again the minimum is $> l$.

In general, a lengthscale can be approximated using binary search. More concretely, the estimate is close when the original lengthscale is long: The difference to the original lengthscale is then between $0.006l$ and $0.008l$. This corresponds to wrongly estimating the largest lengthscale of SVHN by 1.28 (17.28 instead of 16.0) or the smallest (Bank) by 0.16 (estimating 2.16 instead of 2.0). For a short lengthscale, the estimate for MNIST91's lengthscale is around 1.04 instead of 1. For cases except MNIST91, the estimate is inaccurate or indeterminable.

Setting (kernel). The goal of the attacker is to determine the kernel used in a black-box GPC. We assume the victim uses one of the following kernels, RBF (with the same lengthscales as before), linear, or polynomial. We exclude the results on Drebin with the linear and polynomial kernel, as their accuracies are close to a random guess.

Attack description (kernel). An RBF-kernel will output close to zero far away from seen data. Hence, we input the target GPC a zero and ones sample and deduce an RBF-kernel is used if the output is close to 0.5. We also use a more extreme, easy to defeat variant of this attack where the given samples contain only features equivalent to ± 10. To preserve feature meaning, we could also compute an *unusual*, far away sample. Due to the diversity of our data-sets, we leave this variant for future work.

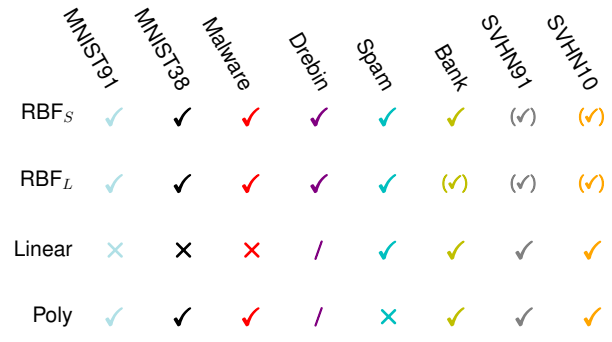


Figure 7.5: Stealing the kernel of a GPC (columns), ✓denotes successful extraction. × denotes a failed attack, (✓) that the attack succeeded only in an easy-to-defeat variant. Some cases were not evaluated (/) as test accuracy was too low.

In case neither output is 0.5, we run a second round of queries. We assume a linear kernel is limited in its expressivity, and leads to less confidently classified data. We thus submit a batch of test points, and classify a kernel as polynomial (nonlinear) if the distribution of outputs is bimodal with most values scattered around 0 and 1. We hence compute the mean of the values above and below 0.5. The threshold for the decision is that both means are further apart than 0.7. This threshold was determined on the additional credit data-set [72], which is otherwise not used in this evaluation.

We train different GPCs, each using a different kernel (RBF kernel with several learned lengthscales, linear, polynomial kernel). The attacker determines, with the above heuristics, the used kernel. Our results are depicted in Figure 7.5.

Results (kernel). In the majority of cases, the attack succeeds independent of lengthscale or kernel used. In three of eight cases, the linear kernel is wrongly determined as nonlinear, indicating that it confidently classified the data against our expectation. In one case, on the spam data, the polynomial kernel is wrongly determined as RBF kernel. There are also some cases on the Bank and SVHN data-sets where the RBF kernel is only correctly predicted if we use the ± 10 -filled samples. Otherwise, the attacker's classification is that the victim uses the polynomial kernel on bank, or the linear kernels on the SVHN tasks.

There are very few differences between using full test data, 500, 50, or as few as 10 samples for the linear/nonlinear query. Only on the Bank data-set the linear kernel was classified as polynomial kernel using 50 samples or less. All other results remained consistent.

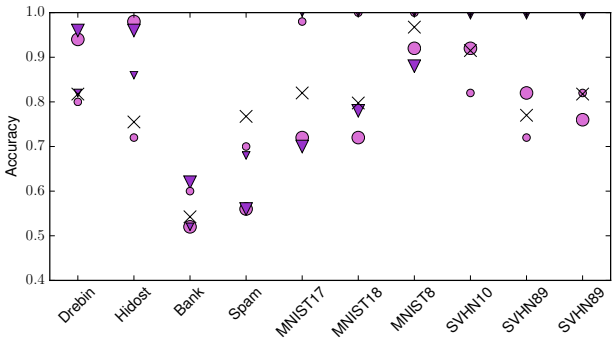
Conclusion. Empirically, the lengthscale can be recovered easily if the training data is (partially) known. This relates to the GP being deterministic. Otherwise, the attacker can reasonably well approximate the lengthscale given that the targeted GPC has a long lengthscale. We hypothesize that the long lengthscale is easier to extract as it is less prone to small changes in the data distribution. The kernel is, instead, easy to

deduce independent of the lengthscale. Our attack currently fails if the linear kernel fits the data well (MNIST, Malware) or the polynomial kernel's decision boundary passes the origin or the ones vector. With a long lengthscale, the RBF kernel (Bank, SVHN) outputs relatively large values even far away from the data. Yet, we find no absolute value for this to happen. Another natural defense to our attack are custom-based kernels. We leave this cases for future work, and conclude that our heuristic works well for the given data-sets and the kernel set {RBF, linear, polynomial}.

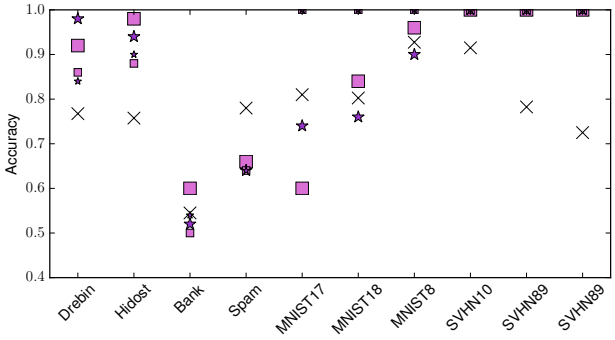
Membership inference

We investigate how good an attacker can determine which points were used in training. First, we study the general setting. Afterwards, we investigate particular settings influencing the attackers success: overfitting, distribution drift, and sparsity.

To study a worst case scenario, the attacker has an oracle that labels a large fraction of the training data as such. This attacker is slightly stronger than the shadow models used in [118]. The attacker trains a fresh classifier that predicts membership for unseen data points.



(a) Classifier trained on variance (triangles) or mean (dots).



(b) Training on mean and variance (square) or latent mean (star).

Figure 7.6: Lengthscale and membership inference on GPC. Bigger symbols denote a long, small symbols a short lengthscale of targeted GPC. x denotes random guess.

Setting. The target GPs are trained using the same lengthscales as before. We then build a data-set using the output of the GPs and membership labels indicating if a data point was used in training. The data-set is split randomly in test data (50 points) and training data (the remainder). The training data is used to train a fresh classifier. We tested DNN, decision trees, random forests and AdaBoost classifiers. As the random forest classifier performed consistently best, we applied only random forests. We report accuracy and random guess on the test data.

Results. We train the random forests on predictive mean (dots), variance (triangle) (Figure 7.6a), mean and variance (squares), or the unnormalized, latent mean (stars) (Figure 7.6b). Overall, using only the predictive mean and a long lengthscale (larger

markers), no data set is vulnerable, with the exception of the two Malware data sets. For mean and variance and latent mean settings, the attacker succeeds in both cases on all SVHN tasks or when using a small lengthscale, with the exception of non-vision tasks. The attacker is also successful on the Malware data sets with a long lengthscale.

On the bank and spam data, the attack is never successful. In general, a shorter lengthscale is more vulnerable. On the Malware data sets, the inverse holds: here, a short lengthscale benefits the defender. Before we focus on these cases, however, we investigate what enables the attacks on the SVHN data and why a short lengthscale is beneficial for the adversary.

Overfitting, distribution drift, and sparsity. We compare training and test accuracies to measure **overfitting**. On the bank data, training and test accuracy are both 100%. On all other data-sets, the difference between test and train accuracy is smaller for a long lengthscale. Hence, slight overfitting occurs at short lengthscales, and enables membership inference.

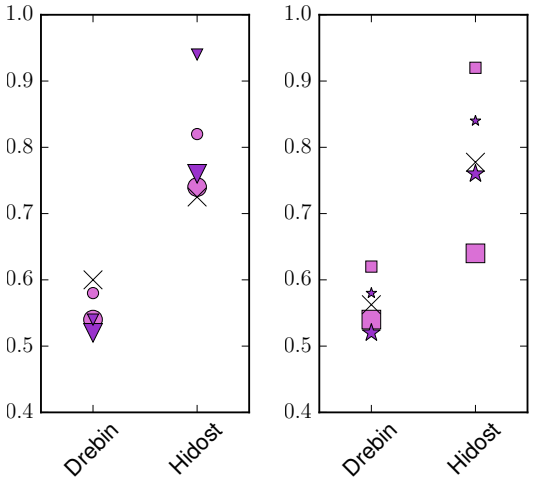


Figure 7.7: Accuracy of membership inference on a sparse GPC. Bigger symbols denote a long, small symbols a short lengthscale on targeted GPC. x denotes random guess. Left plot: Classifier trained on variance (triangles) or mean (dots). Right: Training on mean and variance (square) or latent mean (star).

To analyze **distribution drift**, we measure the standard deviation over the distances between training and test data. As GP adapts the similarity during training, we expect the test data to cause larger variance in the distance if the data is distributed differently. All SVHN and the MNIST8 settings with a small lengthscale show a variance two magnitudes larger between test and training data than among either. Thus, the attack was enabled as training and test data were different from the perspective of GPC. This might imply that the model is not expressive enough to model the data in detail.

Two cases of successful membership inference are left unexplained: the Malware data-sets, Hidost and Drebin. We suspect that **sparsity** causes the vulnerability. The average percentage of features > 0 on the full data-set is $< 0.001\% \pm 0.0006$ on Drebin and $\sim 12\% \pm 3.8$ on Hidost. Next is MNIST (1 vs 7 with around $14\% \pm 4.1$, 1 vs 8 with $\sim 16\% \pm 6.5$ and 8 with $\sim 18\% \pm 5.2$). All other data-sets exhibit less sparsity ($> 20\%$, Spam) or well above 70% (all remaining data-sets).

The difference in sparsity between Hidost and MNIST is small, yet the discrepancy to robust data-sets (Bank, Spam) is large. To account for sparsity, we apply a GPC using inducing variables (GPy's sparse GPC). Such a GPC also optimizes over the training points: the training data is then not directly stored.

In Figure 7.7, we investigate the same settings from the previous study. The

attacker's accuracy is now on all settings close to a random guess, with the exception of a short lengthscale for Hidost on mean or variance, latent mean, or mean and variance. For Drebin, a very small improvement over random guess occurs when a short lengthscale is used and the attacker accesses the GP's predictive mean and variance.

Conclusion. Even under a strong attacker, membership inference is not successful when there is no distribution drift, overfitting is properly taken care of, or a sparse GP with a long lengthscale is applied. Robustness of GPs towards membership inference is somewhat expected, as a GP is not required to be overly confident on training data. The effect of the lengthscale is also intuitive. A short lengthscale allows each training point only local influence, easing inference about membership. With a long lengthscale, each point influences a large area, making it harder to locate the exact training point.

Conclusion

Previous chapters have shown that although education is helpful, defending one attack can already be hard. In this section, we showed that in general, attack vectors on classification should not be seen in isolation, as a mitigation towards one attack might enable or ease another attack.

Formally, we show that evasion is enabled by learning, and any learned GP is vulnerable. Against possible intuition, lazy learning is not per se more vulnerable towards IP attacks. Still, a re-computation of the lengthscale is possible if kernel and the training data are fully known. Yet, no further parameters can be analytically retrieved from given output.

We also study empirical vulnerability, and leveraged the property of a GP to fit a model with a predefined decision function curvature. Summarizing, a short lengthscale leaks the data, and is vulnerable to optimized evasion attacks. A long lengthscale leaks the parameters of the GP, and is vulnerable to one-step attacks with large ϵ. The kernel can be determined independent of the used lengthscale.

Although our experiments have been carried out only on GP classifiers, preliminary work suggests that similar trade-offs might exist on other classifiers such as deep neural networks [79]. In this sense, this chapter shows the need of such works, as we conclude that attacks on classification should not be studied in isolation, but in relation to each other.

8

The lottery ticket hypothesis

Introduction

In the previous chapters, we have seen different aspects of ML security, and why obtaining a robust model, in particular against possibly several attacks at once, is hard. In this section we will conclude this thesis with a more optimistic perspective: how understanding ML algorithms better imrpoves our knowledge in AML as well.

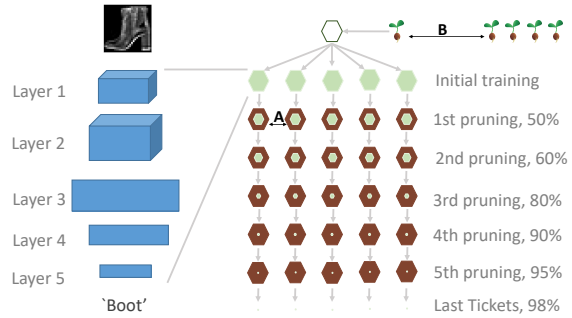
To this end, we will investigate the lottery ticket hypothesis. This hypothesis focuses in the winning subnetwork that emerges from an iterative pruning process. As a security researcher, one might be curious if this winning ticket can be *altered* such that the victim obtains a model that is not optimal anymore. However, it turns out that the iterative pruning procedure effectively picks a new winning subnetwork each training. The overlap between to runs does not exceed what would be expected if the tickets were chosen randomly. Although this knowledge fails to yield an attack, it does increase our knowledge about neural network training and lottery tickets.

Experimental setting

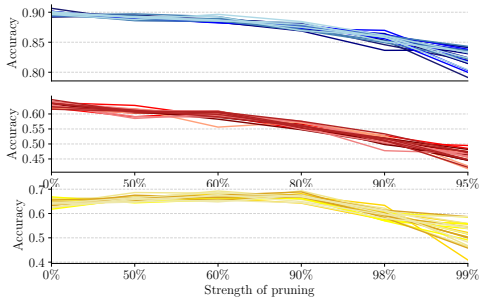
We describe our experimental setting, starting with the network architecture, the details of training and the used baselines. We apply a convolutional network, consisting in two convolutional layers (with 6 and 16 5×5 filters each) and max-pooling. Two dense layers (with 120 and 84 units, respectively) follow before the softmax output. We further train a small ResNet18 [50] to verify that our results hold independent of model size. We generally plot all layers of the small networks and chose randomly five layers of the ResNet (1,11,12,18, and 19) for visualizations. Additional layers can be plotted using the supplementary material.

Each network is trained for 15 epochs—we train few epochs as previous work shows that winning tickets emerge early in training [1, 36, 146]. We further obtain the winning tickets as stated in [147]: given a percentage, we prune this percent of the smallest weights at the end of training.

The experimental set-up is visualized in Figure 8.1a. We choose five initial weight initializations (five random seeds). These five initial weights are kept, however we do not fix any other randomness. On each of the five initial weights, we run five independent iterative pruning procedures. Each of these five runs yields one final ticket in six pruning steps, as visualized in Figure 8.1a. At each pruning step, we prune 50%, 60%, 80%, 90%, 95%, and 98% compared to the size of the initial/original weights. Relative to the kept weights, from first to second step, 20% more weights are pruned. We prune slightly more weights for the ResNet, using percentages 50%, 60%, 90%, 98%, 99%, and 99.9%. The reason is that accuracy is fairly stable for the percentages above, possibly due to the larger weight matrices or more overall weights due to skip connections (see Figure 8.1b). With the given setting, the small networks perform best in pruning steps one, and two, and afterwards decreases with stronger pruning. The ResNet performs best in pruning iteration one, two and three, then, accuracy decreases. We also investigate cases of very small masks with decreased accuracy, as these networks still show good performance relative to their size. For the remainder of the paper, we use the terms mask and ticket interchangeably.



(a) Experimental set-up, example using Fashion MNIST.



(b) Accuracies of pruning steps.

Figure 8.1: (a): Experimental set-up. We fix an architecture and derive initial weights for a seed. These same weights are trained five times, computing tickets in an iterative retraining process. We compare the tickets obtained from one seed (**A**), across seeds (**B**), or across tasks (not visualized). (b): Accuracies for tasks, networks, and pruning steps. From top to bottom: Fashion MNIST, CIFAR10, ResNet18 on CIFAR10. Each line is one run, each color shade one seed. Plots are best seen in color.

The amount of a random chance overlap between two masks is determined using the hyper-geometric distribution. It specifies given a population (size of weight matrix $m \times n$) and a number of objects with a particular feature (e.g., is part of mask M_1) how large the expected overlap is when drawing x new objects (e.g., weights from mask M_2). In other words, in our case the hyper-geometric distribution is parametrized by the number of individual weights mn and the size of the ticket $|t| < mn$. For a number of successes or overlaps $x < mn$, its probability mass function is specified by

$$p(x) = \frac{\binom{|t|}{x}\binom{mn-|t|}{|t|-x}}{\binom{mn}{|t|}} . \quad (8.1)$$

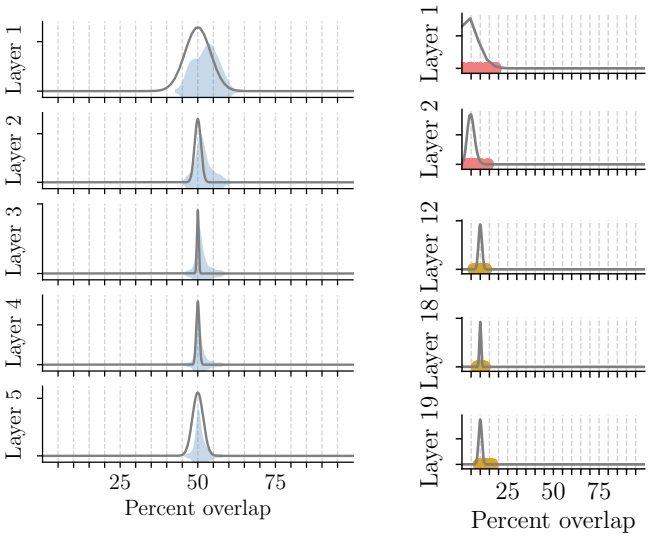


Figure 8.2: Percent overlap between masks across initializations. Left side: Fashion MNIST, first pruning step, as violin plot. Right, upper two: CIFAR10, fifth pruning step. Right, lower three: ResNet on CIFAR10, third pruning step. Gray curve is the hyper-geometric PMF.

In Figure 8.2, we plot the overlap of masks across tasks (denoted as **B** in Figure 8.1a). The observed overlaps (colored area or dots) fall within areas of high probability mass (gray curve in 8.2), confirming our choice. Hence, we depict this baseline in all of the

following plots that show overlaps between tickets.

Experiments

Previous work mentions that there might be several, not one, winning ticket in each network [146]. We confirm in our setting that there are several wining tickets: for each initialization, we run the training-pruning-resetting procedure five times, and measure the distance between the obtained masks. Indeed, the masks are not equivalent, but vary greatly. As they show no overlap beyond chance, we investigate whether there are shared/unused weights beyond chance, or rank correlations between masks. We then check whether the tickets are equivalent under weight-space symmetry. As a final sanity check, we rerun some of the experiments with partially fixed randomness and confirm that these tickets do show overlap beyond the expected baseline.

Are winning tickets unique?

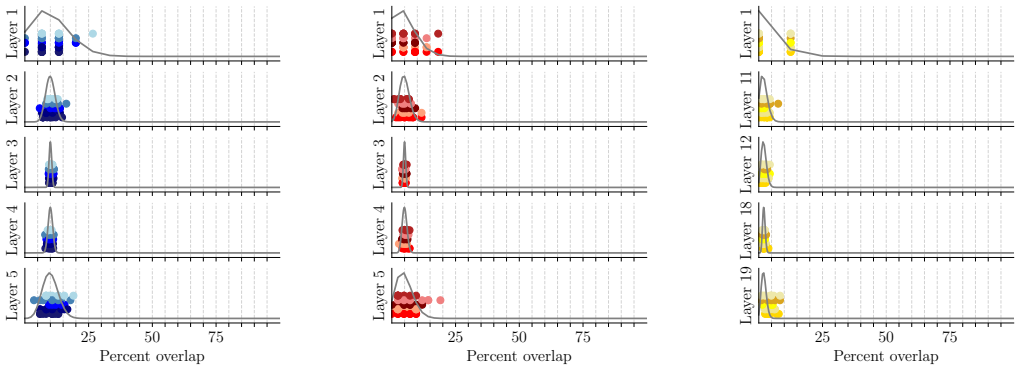
We compare the overlaps between all tickets generated from one initialization (case **A** from Figure 8.1a) after iterative pruning. More concretely, we consider pruning levels of 80% (Fashion MNIST), 95% (CIFAR) and 98% (ResNet, CIFAR). Five training runs are compared among each other, where we count any similarity once, yielding 10 values for each seed. Each experiment contains five seeds or initial weights, yielding 50 similarities in total. We compare the overlaps to the hyper-geometric baseline, which we plot in gray. To ease comparability across layers, all overlaps are shown in percent of the mask size in Figure 8.3.

Results. On Fashion MNIST (Figure 8.3a), the expected overlap lies around 10%. The first layer's overlaps are less than 20%, with some overlaps at 26%. The second layer's overlaps range between 5% and 15%. The third layer's overlaps are scattered with low variance around 10%. Analogously, the fourth layer's overlaps show little variance, and are scattered around 10%. The last layer's overlaps vary between 3% and 20%.

On CIFAR (Figure 8.3b), we investigate a smaller ticket, sized 10% of the original weights. The expected overlaps vary around 5%. In particular the first layer exhibits overlaps smaller than 20%. The second layer's overlaps also match the baseline, varying between 0% and 12%. The overlaps of the third and fourth layer are scattered closely around 5%, where the fourth layer shows (as expected) slightly higher variance. In the last layer, overlaps lie between 0% and 15%, with an outlier at 19%.

The expected overlaps of ResNet are around 2.5%, as we consider a small ticket of only 10% of the original weights. The first layer exhibits, as expected, overlaps between 0% and 12.5%. The eleventh, twelfth and eighteenth layers overlaps vary between 0% and 5%. In the second layer, some outliers show overlaps of 7%. The last layer shows slightly higher overlaps, as expected, of up to 10%.

Conclusion. Tickets for the same initialization are not equivalent, and show barely more overlap than expected when we consider the hyper-geometric distribution as a baseline.



(a) Fashion MNIST, 80% of weights pruned in 3 iterations.

(b) CIFAR10, 90% of weights pruned in 4 iterations.

(c) CIFAR10, 90% of weights pruned in 3 iterations.

Figure 8.3: Percentage of overlap between pruned masks (pruning after 15 epochs) of 5 runs, each using 5 initializations. Each initialization is one shade, y position is based on seed and carries no further meaning. The gray curve denotes the PMF of overlaps given the hyper geometric distribution.

Are there similarities beyond overlap between tickets?

To verify that there are not structural similarities that we missed in the previous experiments, we investigate how much individual tickets for one initialization vary, and how unique they are. In this step, we again compare within one initialization (**A** in Figure 8.1a). To compare uniqueness, we compute for each initialization how many weights are contained in all five masks obtained from randomized training. Further, we investigate the inverse question: how many weights are always pruned, and never form part of a mask. To capture another form of relationship, we plot the rank correlation between initial and final weights. We then draw an overall conclusion.

How many weights are shared?

We first plot the weights forming part of all five masks for one initialization. To ease understanding, we normalize the number of weights and plot percentages in Figure 8.4a. 100% denote the maximal number of weights that can be shared, e.g. 50% of the weights for pruning step one, 40% for pruning step two, etc. Since we compare repeated trials (overlap between several masks), the hyper-geometric is not a valid baseline.

We now derive an approximation to compute the overlap of repeated trials. Consider that we cannot use the binomials, or other distribution with replacement here. The hyper-geometric from the background section, however, does not allow for repeated trials with replacement. We hence approximate the baseline based on the hyper-geometric. First, we define the population size (size of the weight matrix) mn, the size of the successes τ and the number of draws $|t|$. Initially and in the main paper, $|t| = \tau$. We

now recap the mean and variance of the hyper-geometric given as

$$\mu = |t|\frac{|t|}{mn} \quad \text{and} \quad \sigma = |t|\frac{|t|}{mn}\frac{mn-|t|}{mn}\frac{mn-|t|}{mn-1} \ . \tag{8.2}$$

Our main task is to compute an approximation for the number of all drawn weights. We first compute τ as only overlap among tickets. For the second drawn ticket, the expected overlap with the first is

$$\tau_1 = |t|\frac{|t|}{mn} \pm |t|\frac{mn-|t|}{mn}\frac{mn-|t|}{mn-1} \ . \tag{8.3}$$

We focus on the average overlap first, and then derive the variance. For the next ticket, we write the probability that it overlaps with the weights that were shared by all previous tickets. Hence, the mean for τ_i with $i > 1$ is

$$\tau_i = |t|\frac{\tau_{i-1}}{mn} \ . \tag{8.4}$$

Since we run 5 independent runs, we need K_4 ($5-1$, as the first ticket does not overlap) in this case.

We now approximate the variance. After [101], $3\times$ the variance of uni-modal distributions contains roughly 95% of the distribution's mass. We hence additionally compute the overlap with $3\times$ the hyper-geometric's variance,

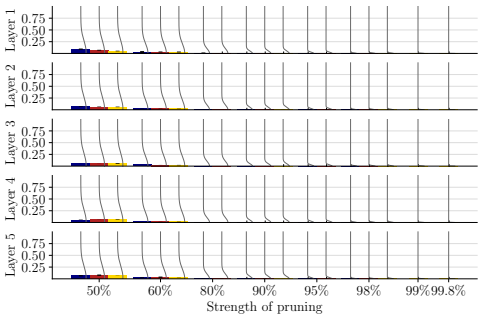
$$\tau_i^{\max} = |t|\frac{\tau_{i-1}^{\max}}{mn} + 3|t|\frac{mn-\tau_{i-1}^{\max}}{mn}\frac{mn-|t|}{mn-1} \ . \tag{8.5}$$

The difference between $\tau_i^{\max}$ and τ_i can then be used to approximate the variance of the underlying distribution. Since we do not know the shape of the true distribution, we will use the estimated mean and variance as parameters for a normal distribution. As however 95% of the mass are contained in $2\times$ the variance of the normal, we use $0.5 \times (\tau_i^{\max} - \tau_i)$ as variance to preserve the ratios.

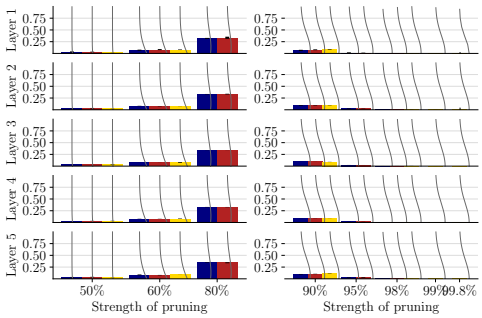
Results. All networks generally share roughly the same, low amount of weights. The amount of shared weights decreases with higher pruning rate, as predicted by our baseline. The shared weights also slightly decrease as we go deeper into the network and consider later layers. The standard deviation between different runs is generally low.

The Fashion MNIST tickets exhibit the highest overlap (10% shared weights) in the first layer at lowest pruning level (50%). Both inner and later layers show slightly lower percentages of shared weights, roughly around 7%. Analogously, the number of shared weights decreases to 0 in the third pruning iteration. The standard deviation between the different runs is overall very small.

The shared weights for CIFAR10 are analogous, and decreases for the first layer as we iterate pruning. Here as well, 0% shared weights are reached at pruning iteration three. The initial overlap is slightly lower than Fashion MNIST, and lies around 8%. Going though the network, in this case, seems not to affect the amount of shared weights. At the second pruning iteration, the overlaps are very low. As for Fashion MNIST, the variance of the shared weights across runs is small.



(a) Percent of weights in all masks.



(b) Percent of weights never in any mask.

Figure 8.4: Influence of the initial weights on the tickets. Blue is Fashion MNIST, red CIFAR, yellow ResNet on CIFAR, and black denotes the standard deviation between the five runs. Gray denotes an approximated baseline. As before, the plotted layers for ResNet are first, 11th, 12th, 18th, and 19th. (a) Percentage of weights shared by all five obtained tickets (for one initialization, normalized by mask size). (b) Percentage of weights not contained in neither five tickets (for one initialization). Right: The left half of the plot is normalized by layer size. The right half is normalized by the maximal disjoint coverage of the masks (e.g., for pruning level 98: $2 \times 5 = 10\%$).

On ResNet, the results are similar to the two smaller networks. The shared weights do not change for the first pruning iteration when going deeper into the network. The first layer reaches zero in pruning iteration three. For deeper layers, the are no shared weights already in iteration two (layer 18, 19).

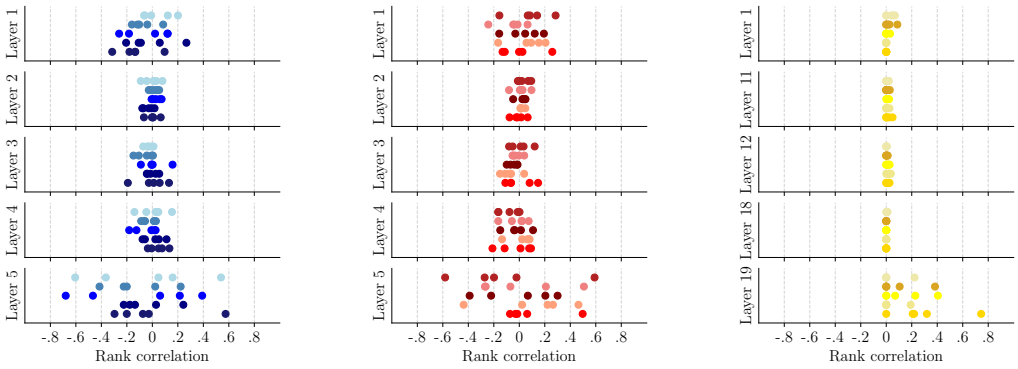
How many weights are left out?

We now investigate which weights never form part of any ticket. For smaller mask sizes (pruning level $> 80\%$), it is impossible that the five masks cover all weights. We thus normalize by the maximal coverage of all five masks or layer size, whichever is smaller. The results are plotted in Figure 8.4b. As in the previous case, we need a baseline. However, the hyper-geometric only determines the overlap between tickets. We want to compute the weights that are in at least one ticket. By the complementary probability, we then obtain the number of weights that are in no ticket. The subsequent derivation differs only slightly from equation 8.4. To obtain all weights that are covered, we add the overlaps and also remaining, not overlapping weights. The first ticket counts fully, hence $\tau_0 = |t|$. For $i > 0$, we write as above

$$\tau_i = (|t| - |t|\frac{\tau_{i-1}}{mn}) \ , \tag{8.6}$$

with the difference that we subtract the overlap and need to compute $\sum \tau_i$. The number of weights not covered on average by any ticket is then $mn - \sum \tau_i$.

Analogous to the previous section, we approximate the spread by computing the maximal intersection and use this value to estimate the variance. After [101], 95% of



(a) Fashion MNIST, 80% of weights pruned in 3 iteration.

(b) CIFAR10, 90% of weights pruned in 4 iterations.

(c) CIFAR10, 90% of weights pruned in 3 iterations.

Figure 8.5: Rank correlation between initial weights and weights in masks (pruning after 15 epochs), each using 5 initializations. Each initialization is one shade, y position is based on seed and carries no further meaning.

any uni-modal distribution is entailed in $3 \times \sigma$. We hence consider the largest overlap as

$$\tau_i^{\max} = |t| - |t|\frac{\tau_{i-1}^{\max}}{mn} + 3|t|\frac{mn - \tau_{i-1}^{\max}}{mn}\frac{mn - |t|}{mn - 1} \; . \tag{8.7}$$

and obtain the largest covered weights as $\sum \tau_i^{\max}$. The difference between these two terms leads to an approximation of the real variance. To preserve the interpretation of the 95% interval, we plot the distribution with mean and set the derived variance as 2σ of a normal distribution.

Results. The amount of weights contained in no ticket is the same for Fashion MNIST, CIFAR10 and ResNet. All values lie within their corresponding expected baselines. In the first pruning step, the amount of not-contained weights is similarly low for all cases, and lies around $2 - 3\%$. The amount increases over 10% (pruning 60%) to 30% when pruning 80% of the weights. For smaller tickets, when seen in relation to the area possibly covered, the percentages decrease again. There are no differences for any network when considering inner or later layers.

How much do the initial weights impact winning tickets?

To investigate another form of potential structure across tickets, we compute the rank correlation between the initial weights and the weights of the resulting winning tickets (setting **A** in Figure 8.1a). A high correlation of 1.0 implies that the order is preserved, -1.0 means the order is inverted, 0 that there is no relationship in terms of rank correlation. We plot the results in Figure 8.5.

Results. The Fashion MNIST correlations are centered for all layers around 0, with differing variances. The first layer's correlations lie between -0.4 and 0.2. The correlations in the second layer range around -0.05 and 0.05. The third and fourth layer exhibit correlations between -0.2 and 0.2. The last layers correlations vary greatly between -0.7 and 0.6.

The small network on CIFAR ten shows, albeit for a smaller ticket, the same pattern as the Fashion MNIST network. The first layer varies between -0.3 and 0.4. The second, third, and fourth layer show little variance, with correlations scattered between -0.2 and 0.2. The last layer show larger variation in correlations, which exhibit values between -0.6 and 0.6.

In contrast to the previous two small networks, the ResNet shows almost no variance in the correlations, which are all scattered with very low variance around 0. An exception are the first, and in particular the last layer. The first layer shows slight positive correlations which are smaller than 0.1. The eleventh, twelfth and eighteenth layer exhibit correlations around zero with low spread. The last layer's correlations, however, lie between 0 and 0.4, with an outlier at 0.75.

Conclusion There is no significant amount of shared weights, or weights that never form part of any mask. The rank correlations between initial weights and final weights for small tickets vary and are centered around 0. First and last layer of the small networks show larger variance in correlations. The ResNet shows less variance of correlations, with the exception of the first and the last layer. In general for ResNet, however, correlations are zero or positive, never negative.

Are tickets variations over the same network?

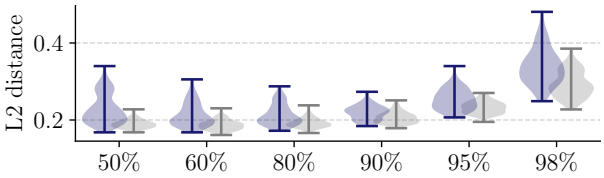
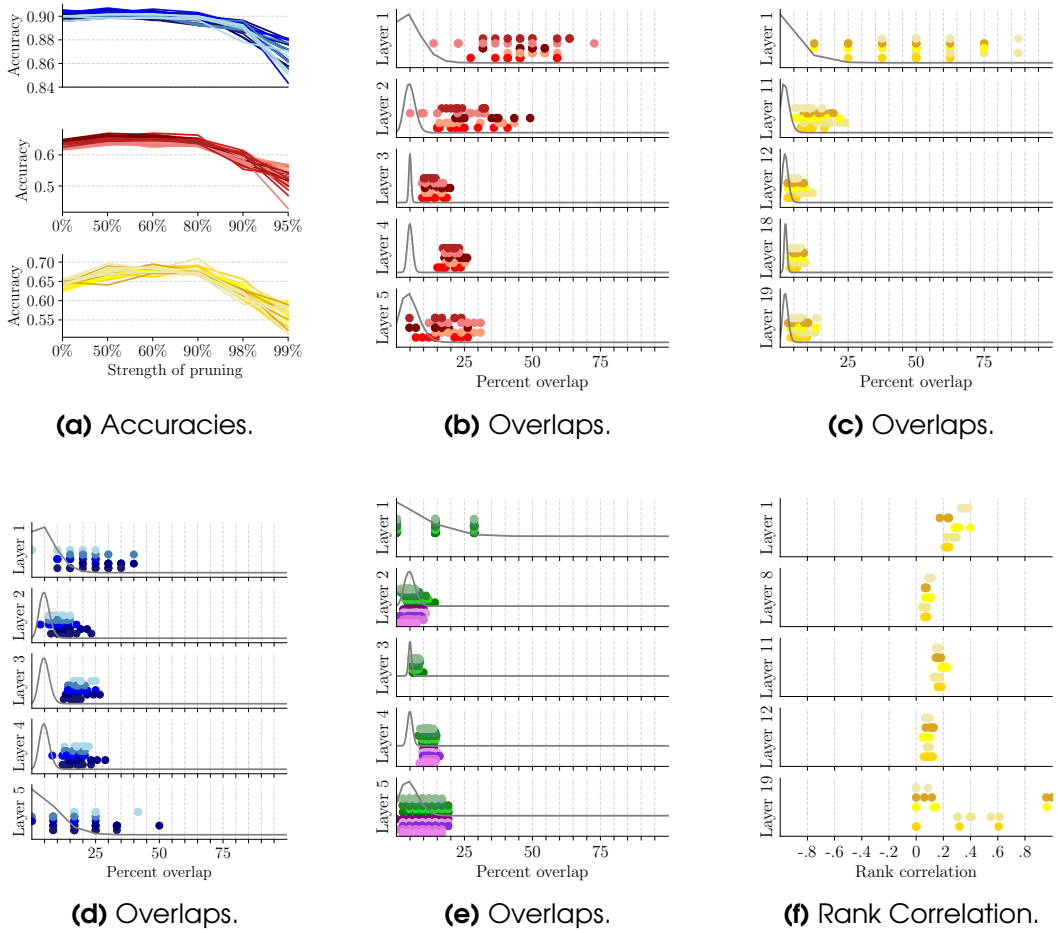


Figure 8.6: CKA similarity between winning tickets within (blue, left) and across (gray, right) seeds for Fashion MNIST.

One might be tempted to explain the differences in tickets by weight-space symmetry: The differing networks would then, in fact, be just variants of the same network. To show that this is not the case, we take advantage that networks, if equivalent in weight-space, will yield the same output (as the variations do not touch functionality). We computed the L_2-distance and depict it on the Fashion MNIST network, where outputs are most similar. We plot the distances of outputs of tickets in for one seed (blue, **A** in Figure 8.1a) as well as distance among seeds (gray, **B** in Figure 8.1a) in Figure 8.6.

Results. There are not two tickets yielding the exact same input.The distances between tickets first decreases as the pruning level increases, then decreases, somewhat similar to the accuracy. The distances across seeds remain stable for the first three pruning iterations, and then increase. The distances across seeds are overall lower than within seeds.

Conclusion. The distance between the outputs increases as we prune iteratively and harvest smaller tickets. As the distance is never zero, we can refute the hypothesis that different tickets are instances of the same network under the weight-space hypothesis.



(a) Accuracies. **(b)** Overlaps. **(c)** Overlaps.

(d) Overlaps. **(e)** Overlaps. **(f)** Rank Correlation.

Figure 8.7: Repeating the previous experiments with slightly fixed randomness. (a): accuracies of different pruning stages. (b): overlaps between CIFAR tickets in the fourth pruning iteration. (c): overlaps between ResNet masks in the third pruning iteration. (d): overlaps between networks on Fashion MNIST with equal-sized layers. (e): overlaps across tasks with same initialization at pruning level 4: green denotes overlap between Fashion MNIST and MNIST, purple between CIFAR and Fashion MNIST. (b) rank correlations on a CIFAR10 ResNet at pruning iteration three.

What is the effect of constraining randomness?

So far, we did not fix randomness at all. Generally, randomness is removed from experiments to increase reproducibility. In our setting, with no randomness at all, we expect the tickets to overlap perfectly. We are thus interested in the gray zone, where randomness is decreased, but not entirely fixed. To this end, we fix randomness in the

batches, but leave the randomness in the gradients. No data augmentation is used. We repeat all previous experiments, and depict a subset of the results in Figure 8.7.

Results – accuracies. Figure 8.7a shows that accuracies are overall higher, and now increase for initial pruning iterations of small networks. Instead of a steady incline, the accuracies increase at the first pruning step, and decrease later, after the third pruning step. This holds irrespective of task or architecture, and is similar to the ResNet without fixed randomness.

Results – overlaps. Also, as expected, the overlaps between tickets are now, due to decreased randomness, larger than expected by the hyper-geometric. We show results on a small CIFAR network (see Figure 8.7b) and a ResNet18 (see Figure 8.7c). The overlaps vary in both cases most for the first, second/eleventh and last layer. In general, the spread of the overlaps seems related to the variance of the hyper-geometric baseline. To verify this, we depict in Figure 8.7d a network on Fashion MNIST where all inner layers have approximately the same size (e.g., 2400 or 2500 weights, first layer 400, last layer 250). Indeed, the variances of all overlaps are once again strongly correlated with the baseline. The average overlap seems not to rely on the random baseline, though.

Results – dependence on data. To check how strong the influence of the fixed randomness is, we compare the overlap with fixed randomness across different tasks in Figure 8.7e. As the small CIFAR network, due to larger input dimensionality, can only be compared in layer two, four and five, we train additional networks on the MNIST dataset [69]. We depict the overlap between CIFAR and Fashion MNIST networks in purple, and MNIST and Fashion MNIST in green. The expected overlap is around 5%. For the first layer, the overlap is as expected. The second layer exhibits marginally higher overlaps. The third layer shows slightly higher overlaps, ranging between 5% and 10%. Layer four's overlaps range between 7.5% and 15%, and are clearly distinct from the expected values. The last layer shows again only slightly higher overlap than expected, ranging between 0% and 20%.

Results – rank correlations. Last but not least, the rank correlation is now not centered at or around zero. From Figure 8.7f, we see that all rank correlations are positive. The first layers correlations are between 0.15 and 0.4. The eighth and twelfth layers correlations vary around 0.1. The eleventh layer shows slightly higher correlations which lie between 0.1 and 0.2. The last layer exhibits large variability of the rank correlations, ranging between 0 and 1.0.

Conclusion. Decreasing randomness also decreases the differences between tickets and leads to more overlap between tickets, even if networks are trained on different tasks. The accuracy also benefits from decreased randomness, and is slightly higher.

Conclusion

When different initializations are used, the overlap between masks is predicted by the hyper-geometric distribution. Using the same initial weights without fixed stochasticity corresponds to the hyper-geometric as well. The hyper-geometric assumes that the probability to chose any element (in our case a weight) is equal to choosing any other element (weight). In other words, we find no evidence that there is any meaningful structure in the weights before training has started: the weights in the winning ticket are chosen at random. Only stochasticity is reduced, the overlaps deviate from the hyper-geometric.

When investigating the resulting winning tickets, there are no other correlations, shared or excluded weights beyond chance. The winning tickets are neither instances of the same network under weight-space symmetry. This highlights the initial large space of possible winning tickets. Our findings also offer further explanation to why starting the training for the pruned network not in iteration 0 but slightly later is successful [35, 36]. With a full restart, it is unlikely that the weights move towards the same winning ticket during training.

We are not able to exploit lottery tickets for training time attack, as changing the ticket will simply give rise to another ticket. Yet, winning tickets give us an intuition on why for example transferability for missclassification does not guarantee which class the targeted classifier outputs [76]. Classifiers, even if trained on the same data with the same initial weights, might exhibit different outputs on the same outputs. Although more work is needed to understand how different tickets arise and how they differ in detail, this work illustrates how both AML and ML knowledge benefit from each other.

9
Conclusion

In this thesis, we have shown several aspects that make machine learning security a hard problem. In the first chapter, we showed that evasion as an attack is inherent to classifiers, as the gradients of their surface can be used to determine features influencing classification outcome. The second part of the first chapter showed that also models that seem different (Gaussian process classification and Bayesian neural networks) suffered from transferability of adversarial examples: computing high confidemce, low uncertainty attacks on one classifier yielded working adversarial inputs with similar confidence and uncertainty for the other classifier. This finding can be attributed to the shared training data, and the resulting similar learned features. An attack was then able to succeed, although the original target classifier was never interacted with. Such transferability increases significantly the difficulty to develop defenses against evasion attacks, and contributes to the ongoing arms-race in evasion mitigations.

In the second chapter, we investigated another attack on classification: We showed that backdoors rely on underfitting backdoor features. Furthermore, we found that non-backdoored models contain underfitted sets of features, too. As some mitigations rely conceptually on the backdoors stable (mis)classification, the question is raised how to truly now whether the found pattern was inserted malicously during training or not.

In the third chapter, we introduced a new training-time attack taking advantage of the complexity of deep learning libraries. In contrast to previous attacks, these attacks were recognizable if the user was aware of the corresponding threat. The implications of the chapter are twofold. On the one hand, it underlines the importance to educate users on possible threats and countermeasures. On the other hand, it shows that the full environment of algorithm needs to be taken into account in security. In other workds, more research is needed to understand the full extend of the security vulnerabilities when machine learning is applied.

In the next chapter, we examined several test-time attacks in relation to each other on one algorithm, Gaussian process classification. We saw that hardening the classifier against an individual attack was possible, but led to being vulnerable to another attack instead. It is thus crucial to not consider attacks individually, but in relation to one another when aiming for security. Our work in this chapter was limited to Gaussian Process classification, leaving open the question in which way different attacks affect each other on for example deep learning. Few preliminary works show here that for example robustness towards evasion increases vulnerability to membership inference. Yet, overall understanding in this respect is still limited.

In the last chapter, we studied the lottery ticket hypothesis. We showed that the hyper-geometric is a valid baseline for the overlap of lottery tickets. Under stochasticity, even if the initial weights are the same, the observed overlaps corresponded to this hypergeometric. As the hypergeometric assumes all elements to be independant, we found no significant structure that affected the resulting winning ticket unless stochasticity is reduced. Our findings backed up prior works finding that starting the training for the pruned network not in iteration 0 but slightly later is successful. Our results also highlighted the initial large space of possible winning tickets. Beyond a pure understanding of the training processes of deep learning, this large space is also related to transferability of attacks. In this sense, work that invetigates the rôle of stochasticity in training might thus also help us understand how classifier vary in terms of robustness.

Beyond this thesis, we have to remark that ML, as a new emerging technology, also faces the problem that on the one hand, it provides many new opportunities and chances to deal with existing sociecital problems. Yet, a realistic security assemesment is crucial to not cause harm by applying ML. As we have seen in this thesis, the aspects contributing to security of machine learning are numerous and not necessarily on the same layer of abstraction. Even worse, our understanding of machine learning security is still incomplete, and new threats are still unveiled. An additional hurdle is that we are currently lacking an overview about the attacks that occurr in the wild. The difficulty of collecting such data is rooted in many problems: many systems are proprietary, and only few cases of severe failure are reported. Furthermore, machine learning or artificial intelligence are currently buzzwords. As a consequence, different systems are framed as for example machine learning, and it is hard to asses which concrete threats are applicable in which case.

Summarizing, incorporating and unifying all aspects of machine learning security is still an open problem. On the other hand, however, we also have to acknowledge the positive impact that security of machine learning has had on the field of machine learning. The discovery of for example evasion attacks and other security breaches has largely increased our knowledge of machine learning models and how they work.

List of Figures

List of Tables

List of Algorithms

Bibliography

Author's Papers for this Thesis

[P1] Grosse, Kathrin et al. Adversarial Examples for Malware Detection. In: *ESORICS*. Springer, 2017. ©Springer 2017, 62–79.

[P2] Grosse, Kathrin et al. The Limitations of Model Uncertainty in Adversarial Settings. *BDL@NeurIPS* (2019).

[P3] Grosse, Kathrin et al. A new measure for overfitting and its implications for backdooring of deep learning. *Under submission* (2020).

[P4] Grosse, Kathrin et al. On the security relevance of initial weights in deep neural networks. In: *ICANN*. Springer, 2020. ©Springer, 2020, 3–14.

[P5] Grosse, Kathrin, Smith, Michael T., and Backes, Michael. Killing Four Birds with one Gaussian Process: The relation between different Test-Time Attacks. In: *ICPR*. IEEE, 2020. ©IEEE 2020.

[P6] Grosse, Kathrin and Backes, Michael. How many winning tickets are there in one DNN? In: *SDM*. SIAM, 2021. ©SIAM 2021.

Further references

[1] Achille, Alessandro, Rovere, Matteo, and Soatto, Stefano. Critical learning periods in deep neural networks. *ICLR* (2019).

[2] Arp, Daniel et al. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In: *NDSS*. 2014.

[3] Athalye, Anish, Carlini, Nicholas, and Wagner, David. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. In: *ICML*. 2018.

[4] Ba, Jimmy et al. Generalization of Two-layer Neural Networks: An Asymptotic Viewpoint. In: *ICLR*. 2019.

[5] Backes, Michael, Bugiel, Sven, and Derr, Erik. Reliable third-party library detection in android and its security applications. In: *CCS*. ACM. 2016.

[6] Bekasov, Artur and Murray, Iain. Bayesian Adversarial Spheres: Bayesian Inference and Adversarial Examples in a Noiseless Setting. *Bayesian Deep Learning at NIPS* (2018).

[7] Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5, 2 (1994), 157–166.

[8] Biggio, Battista, Nelson, Blaine, and Laskov, Pavel. Support vector machines under adversarial label noise. In: *Asian conference on machine learning*. 2011, 97–112.

[9] Biggio, Battista and Roli, Fabio. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition* 84 (2018), 317–331.

[10] Biggio, Battista et al. Evasion attacks against machine learning at test time. In: *ECML PKDD*. Springer. 2013.

[11] Bishop, Christopher M. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007.

[12] Blaas, Arno et al. Robustness Quantification for Classification with Gaussian Processes. In: 2020.

[13] Bogunovic, Ilija et al. Adversarially Robust Optimization with Gaussian Processes. In: *NIPS*. 2018.

[14] Bojarski, Mariusz et al. End to end learning for self-driving cars. *NIPS 2016 Deep Learning Symposium* (2016).

[15] Bradshaw, John, Matthews, Alexander G de G, and Ghahramani, Zoubin. Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476* (2017).

[16] Carlini, Nicholas and Wagner, David. Adversarial examples are not easily detected: Bypassing ten detection methods. In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM. 2017, 3–14.

[17] Carlini, Nicholas and Wagner, David. Towards evaluating the robustness of neural networks. In: *2017 IEEE S&P*. IEEE. 2017, 39–57.

[18] Chen, Bryant et al. Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering. In: *Workshop on Artificial Intelligence Safety at AAAI*. 2019.

[19] Chen, Xinyun et al. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[20] Cheney, Nicholas, Schrimpf, Martin, and Kreiman, Gabriel. On the Robustness of Convolutional Neural Networks to Internal Architecture and Weight Perturbations. *arXiv preprint arXiv:1703.08245* (2017).

[21] Chou, Edward et al. SentiNet: Detecting Physical Attacks Against Deep Learning Systems. *arXiv preprint arXiv:1812.00292* (2018).

[22] Cohen, Jeremy M., Rosenfeld, Elan, and Kolter, J. Zico. Certified Adversarial Robustness via Randomized Smoothing. In: *ICML*. 2019, 1310–1320.

[23] Crowley, Elliot J et al. Pruning neural networks: is it time to nip it in the bud? *NIPS 2018 Workshop CDNNRIA* (2018).

[24] Cullina, Daniel, Bhagoji, Arjun Nitin, and Mittal, Prateek. PAC-learning in the presence of adversaries. In: *NIPS*. 2018.

[25] Dahl, George E et al. Large-scale malware classification using random projections and neural networks. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, 3422–3426.

[26] Dalvi, Nilesh et al. Adversarial Classification. In: KDD '04. 2004.

[27] Degwekar, Akshay, Nakkiran, Preetum, and Vaikuntanathan, Vinod. Computational Limitations in Robust Classification and Win-Win Results. In: *COLT*. 2019, 994–1028.

[28] Demontis, Ambra et al. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In: *28th {USENIX} Security Symposium ({USENIX} Security 19)*. 2019, 321–338.

[29] Denoeux, Thierry and Lengellé, Régis. Initializing back propagation networks with prototypes. *Neural Networks* 6, 3 (Jan. 1993). ISSN: 0893-6080.

[30] Desai, Shrey, Zhan, Hongyuan, and Aly, Ahmed. Evaluating Lottery Tickets Under Distributional Shifts. In: *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*. 2019, 153–162.

[31] Drago, G.P. and Ridella, S. Statistically controlled activation weight initialization (SCAWI). *IEEE Transactions on Neural Networks* 3, 4 (July 1992), 627–631. ISSN: 1045-9227.

[32] Ebrahimi, Nader, Soofi, Ehsan S, and Soyer, Refik. Information measures in perspective. *International Statistical Review* 78, 3 (2010), 383–412.

[33] Fligner, Michael A and Killeen, Timothy J. Distribution-free two-sample tests for scale. *Journal of the American Statistical Association* 71, 353 (1976), 210–213.

[34] Frankle, Jonathan and Carbin, Michael. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. In: *ICLR*. 2019.

[35] Frankle, Jonathan et al. Stabilizing the lottery ticket hypothesis. *arXiv preprint arXiv:1903.01611* (2019).

[36] Frankle, Jonathan et al. Linear mode connectivity and the lottery ticket hypothesis. In: *ICML*. 2020.

[37] Frigge, Michael, Hoaglin, David C, and Iglewicz, Boris. Some implementations of the boxplot. *The American Statistician* 43, 1 (1989), 50–54.

[38] Gaier, Adam and Ha, David. Weight Agnostic Neural Networks. *NeurIPS* (2019).

[39] Gal, Yarin and Ghahramani, Zoubin. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In: *ICML*. 2016, 1050–1059.

[40] Giryes, Raja, Sapiro, Guillermo, and Bronstein, Alex M. Deep Neural Networks with Random Gaussian Weights: A Universal Classification Strategy? *IEEE Transactions on Signal Processing* 64, 13 (July 2016), 3444–3457.

[41] Gomez, Aidan N et al. Targeted dropout. *NIPS 2018 Workshop CDNNRIA* (2018).

[42] Gondara, Lovedeep, Wang, Ke, and Carvalho, Ricardo Silva. The Differentially Private Lottery Ticket Mechanism. *arXiv preprint arXiv:2002.11613* (2020).

[43] Goodfellow, Ian J et al. Explaining and Harnessing Adversarial Examples. In: *ICLR*. 2015.

[44] Goodfellow, Ian J., Papernot, Nicolas, and McDaniel, Patrick D. cleverhans v0.1: an adversarial machine learning library. *CoRR* abs/1610.00768 (2016).

[45] GPy. *GPy: A Gaussian process framework in python*. `http://github.com/SheffieldML/GPy`. since 2012.

[46] Gu, Tianyu et al. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access* 7 (2019), 47230–47244.

[47] Hanin, Boris. Which Neural Net Architectures Give Rise to Exploding and Vanishing Gradients? In: *NIPS*. 2018.

[48] Hanin, Boris and Rolnick, David. How to Start Training: The Effect of Initialization and Architecture. In: *NIPS*. 2018.

[49] He, Kaiming et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *ICCV*. 2015.

[50] He, Kaiming et al. Deep residual learning for image recognition. In: *CVPR*. 2016, 770–778.

[51] Hein, Matthias and Andriushchenko, Maksym. Formal guarantees on the robustness of a classifier against adversarial manipulation. In: *NIPS*. 2017, 2266–2276.

[52] Hitaj, Briland, Ateniese, Giuseppe, and Perez-Cruz, Fernando. Deep models under the GAN: information leakage from collaborative deep learning. In: *CCS*. ACM. 2017, 603–618.

[53] Hodges, JL. The significance probability of the Smirnov two-sample test. *Arkiv för Matematik* 3, 5 (1958), 469–486.

[54] James, Gareth M. Variance and bias for general loss functions. *Machine learning* 51, 2 (2003), 115–135.

[55] Javed, Amir, Burnap, Pete, and Rana, Omer. Prediction of drive-by download attacks on Twitter. *Information Processing & Management* 56, 3 (2019), 1133–1145.

[56] Ji, Yujie, Zhang, Xinyang, and Wang, Ting. Backdoor attacks against learning systems. In: *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2017, 1–9.

[57] Jiang, Yiding et al. Fantastic Generalization Measures and Where to Find Them. In: *ICLR*. 2020.

[58] Juuti, Mika et al. PRADA: protecting against DNN model stealing attacks. In: *EuroS&P*. IEEE. 2019, 512–527.

[59] Kadmon, Jonathan and Sompolinsky, Haim. Transition to Chaos in Random Neuronal Networks. *Phys. Rev. X* 5 (4 Nov. 2015), 041030.

[60] Kohavi, Ron, Wolpert, David H, et al. Bias plus variance decomposition for zero-one loss functions. In: *ICML*. Vol. 96. 1996, 275–83.

[61] Kolosnjaji, Bojan et al. Deep learning for classification of malware system call sequences. In: *Australasian Joint Conference on Artificial Intelligence*. Springer. 2016, 137–149.

[62] Kolosnjaji, Bojan et al. Adversarial malware binaries: Evading deep learning for malware detection in executables. In: *2018 26th European Signal Processing Conference (EUSIPCO)*. IEEE. 2018, 533–537.

[63] Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: *NIPS*. 2012.

[64] Krizhevsky, Alex, Hinton, Geoffrey, et al. Learning multiple layers of features from tiny images (2009).

[65] Kumar, Rajesh et al. Effective and explainable detection of Android malware based on machine learning algorithms. In: *Proceedings of the 2018 International Conference on Computing and Artificial Intelligence*. 2018, 35–40.

[66] Laskov, Pavel et al. Practical evasion of a learning-based classifier: A case study. In: *IEEE S&P*. 2014.

[67] Lauinger, Tobias et al. Thou shalt not depend on me: Analysing the use of outdated javascript libraries on the web. In: *NDSS*. 2017.

[68] Le, Van Lam et al. Anatomy of Drive-by Download Attack. In: *Eleventh Australasian Information Security Conference, AISC 2013*. 2013, 49–58.

[69] LeCun, Yann et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* (1998).

[70] Lee, Guang-He et al. Tight certificates of adversarial robustness for randomly smoothed classifiers. In: *NeurIPS*. 2019.

[71] Li, Yingzhen and Gal, Yarin. Dropout Inference in Bayesian Neural Networks with Alpha-divergences. In: *ICML*. 2017.

[72] Lichman, M. *UCI Machine Learning Repository*. 2013. URL: `http://archive.ics.uci.edu/ml`.

[73] Ling, Xiang et al. Deepsec: A uniform platform for security analysis of deep learning model. In: *IEEE S&P*. 2019.

[74] Liu, Shengchao, Papailiopoulos, Dimitris, and Achlioptas, Dimitris. Bad Global Minima Exist and SGD Can Reach Them. *ICML 2019 Workshop Deep Phenomena* (2019).

[75] Liu, Yannan et al. Fault injection attack on deep neural network. In: *Proceedings of the 36th International Conference on Computer-Aided Design*. IEEE Press. 2017, 131–138.

[76] Liu, Yanpei et al. Delving into Transferable Adversarial Examples and Black-box Attacks. In: *ICLR*. 2017.

[77] Liu, Yingqi et al. Trojaning Attack on Neural Networks. In: *NDSS*. 2018.

[78] Liu, Yingqi et al. ABS: Scanning neural networks for back-doors by artificial brain stimulation. In: *CCS*. 2019.

[79] Liwei Song, Reza Shokri and Mittal, Prateek. Membership Inference Attacks against Adversarially Robust Deep Learning Models. In: *2nd Deep Learning and Security Workshop*. 2019.

[80] Maiorca, Davide, Corona, Igino, and Giacinto, Giorgio. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious pdf files detection. In: *CCS*. 2013.

[81] Malach, Eran et al. Proving the Lottery Ticket Hypothesis: Pruning is All You Need. In: *ICML*. 2020.

[82] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[83] Mei, Song and Montanari, Andrea. The generalization error of random features regression: Precise asymptotics and double descent curve. *arXiv preprint arXiv:1908.05355* (2019).

[84] Melis, Marco et al. Is Deep Learning Safe for Robot Vision? Adversarial Examples Against the iCub Humanoid. In: *IEEE ICCV Workshops 2017*. 2017, 751–759.

[85] Mika, S. et al. Fisher discriminant analysis with kernels. In: *IEEE Signal Processing Society Workshop (Cat. No.98TH8468)*. Aug. 1999, 41–48.

[86] Miyato, Takeru et al. Distributional Smoothing by Virtual Adversarial Examples. In: *ICLR*. 2016.

[87] Moosavi-Dezfooli, Seyed-Mohsen et al. Universal adversarial perturbations. In: *CVPR*. 2017.

[88] Morcos, Ari S et al. One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. *NeurIPS* (2019).

[89] Nelson, Blaine et al. Query strategies for evading convex-inducing classifiers. *The Journal of Machine Learning Research* 13, 1 (2012), 1293–1332.

[90] Netzer, Yuval et al. Reading Digits in Natural Images with Unsupervised Feature Learning. In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011.

[91] Novak, Roman et al. Sensitivity and Generalization in Neural Networks: an Empirical Study. In: *ICLR*. 2018.

[92] Oh, Seong Joon, Schiele, Bernt, and Fritz, Mario. Towards reverse-engineering black-box neural networks. In: *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 2019, 121–144.

[93] Olkin, Ingram. *Contributions to probability and statistics: essays in honor of Harold Hotelling*. Stanford University Press, 1960.

[94] Onishi, Tadashi et al. End-to-end Learning Method for Self-Driving Cars with Trajectory Recovery Using a Path-following Function. In: *IJCNN*. IEEE, 2019, 1–8.

[95] Papernot, Nicolas. A Marauder's Map of Security and Privacy in Machine Learning. *arXiv preprint arXiv:1811.01134* (2018).

[96] Papernot, Nicolas, McDaniel, Patrick, and Goodfellow, Ian J. Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples. *arXiv preprint arXiv:1605.07277* (2016).

[97] Papernot, Nicolas et al. The Limitations of Deep Learning in Adversarial Settings. In: *EuroS&P*. 2016.

[98] Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In: *ICML*. 2013, 1310–1318.

[99] Pierazzi, Fabio et al. Intriguing properties of adversarial ML attacks in the problem space. In: *IEEE S&P*. IEEE. 2020, 1332–1349.

[100] Poole, Ben et al. Exponential expressivity in deep neural networks through transient chaos. In: *NeurIPS*. 2016.

[101] Pukelsheim, Friedrich. The three sigma rule. *The American Statistician* 48, 2 (1994), 88–91.

[102] Raghunathan, Aditi, Steinhardt, Jacob, and Liang, Percy. Certified Defenses against Adversarial Examples. In: *ICLR*. 2018.

[103] Ramanujan, Vivek et al. What's Hidden in a Randomly Weighted Neural Network? In: *CVPR*. 2019.

[104] Rasmussen, Carl Edward and Williams, Christopher K. I. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006.

[105] Rawat, A., Wistuba, M., and Nicolae, M.-I. Adversarial Phenomenon in the Eyes of Bayesian Deep Learning. *ArXiv e-prints* abs/1711.08244 (Nov. 2017).

[106] Remaki, L. and Cheriet, M. KCS-new kernel family with compact support in scale space: formulation and impact. *IEEE Transactions on Image Processing* 9, 6 (2000), 970–981.

[107] Rieck, Konrad et al. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 4 (2011), 639–668.

[108] Rozsa, Andras, Günther, Manuel, and Boult, Terrance E. Are accuracy and robustness correlated? In: *Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*. IEEE. 2016, 227–232.

[109] Rubinstein, Benjamin IP et al. Antidote: understanding and defending against poisoning of anomaly detectors. In: *ACM SIGCOMM Conference on Internet Measurement*. ACM. 2009, 1–14.

[110] Russu, Paolo et al. Secure Kernel Machines against Evasion Attacks. In: *AISec@CCS*. ACM, 2016, 59–69.

[111] Salem, Ahmed et al. ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models. In: *NDSS*. 2019.

[112] Salem, Ahmed et al. Updates-leak: Data set inference and reconstruction attacks in online learning. In: *USENIX*. 2020.

[113] Saxe, Joshua and Berlin, Konstantin. Deep neural network based malware detection using two dimensional binary program features. In: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. IEEE. 2015, 11–20.

[114] Sayfullina, Luiza et al. Efficient Detection of Zero-day Android Malware Using Normalized Bernoulli Naive Bayes. In: *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. 2015, 198–205.

[115] Scheirer, Walter J., Jain, Lalit P., and Boult, Terrance E. Probability Models for Open Set Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 11 (2014), 2317–2324.

[116] Shabtai, Asaf, Fledel, Yuval, and Elovici, Yuval. Automated static code analysis for classifying android applications using machine learning. In: *2010 international conference on computational intelligence and security*. IEEE. 2010, 329–333.

[117] Sharma, Yash, Ding, Gavin Weiguang, and Brubaker, Marcus A. On the Effectiveness of Low Frequency Perturbations. In: *IJCAI*. 2019, 3389–3396.

[118] Shokri, Reza et al. Membership inference attacks against machine learning models. In: *IEEE S&P*. IEEE. 2017, 3–18.

[119] Shu, Hai and Zhu, Hongtu. Sensitivity analysis of deep neural networks. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 2019, 4943–4950.

[120] Silver, David et al. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[121] Smith, Lewis and Gal, Yarin. Understanding Measures of Uncertainty for Adversarial Example Detection. In: *UAI*. 2018.

[122] Smith, Michael Thomas et al. Differentially Private Regression with Gaussian Processes. In: *AISTATS*. 2018, 1195–1203.

[123] Šrndić, Nedim and Laskov, Pavel. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security* 2016 (Sept. 2016).

[124] Stevens, David and Lowd, Daniel. On the hardness of evading combinations of linear classifiers. In: *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. 2013, 77–86.

[125] Stevens, R. et al. Summoning Demons: The Pursuit of Exploitable Bugs in Machine Learning. *CoRR* abs/1701.04739 (2017).

[126] Suciu, Octavian et al. When Does Machine Learning FAIL? Generalized Transferability for Evasion and Poisoning Attacks. In: *USENIX*. 2018, 1299–1316.

[127] Szegedy, Christian et al. Intriguing properties of neural networks. *ICLR* (2014).

[128] Tan, Te Juin Lester and Shokri, Reza. Bypassing Backdoor Detection Algorithms in Deep Learning. In: *Euro S&P*. 2020.

[129] Tanay, Thomas and Griffin, Lewis. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690* (2016).

[130] Tramèr, Florian et al. Stealing machine learning models via prediction apis. In: *USENIX*. 2016, 601–618.

[131] Tramèr, Florian et al. On adaptive attacks to adversarial example defenses. *arXiv preprint arXiv:2002.08347* (2020).

[132] Tran, Brandon, Li, Jerry, and Madry, Aleksander. Spectral signatures in backdoor attacks. In: *NIPS*. 2018.

[133] Vialatte, Jean-Charles and Leduc-Primeau, François. A study of deep learning robustness against computation failures. *arXiv preprint arXiv:1704.05396* (2017).

[134] Vinyals, Oriol et al. Matching networks for one shot learning. In: *NIPS*. 2016, 3630–3638.

[135] Wang, Bolun et al. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In: *IEEE S&P*. IEEE, 2019.

[136] Wang, Yizhen, Jha, Somesh, and Chaudhuri, Kamalika. Analyzing the Robustness of Nearest Neighbors to Adversarial Examples. In: *ICML*. 2018, 5120–5129.

[137] Werpachowski, Roman, György, András, and Szepesvári, Csaba. Detecting overfitting via adversarial examples. In: *NeurIPS*. 2019.

[138] Xiang, Zhen, Miller, David J, and Kesidis, George. Detection of Backdoors in Trained Classifiers Without Access to the Training Set. *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[139] Xiao, Han, Rasul, Kashif, and Vollgraf, Roland. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[140] Xiao, Qixue et al. Wolf in Sheep's Clothing-The Downscaling Attack Against Deep Learning Applications. *CoRR* abs/1712.07805 (2017).

[141] Xiao, Qixue et al. Security Risks in Deep Learning Implementations. In: *2018 IEEE Security and Privacy Workshops, SP Workshops 2018, San Francisco, CA, USA, May 24, 2018*. 2018, 123–128.

[142] Xie, Di, Xiong, Jiang, and Pu, Shiliang. All You Need is Beyond a Good Init: Exploring Better Solution for Training Extremely Deep Convolutional Neural Networks with Orthonormality and Modulation. In: *CVPR*. 2017.

[143] Yam, Jim YF and Chow, Tommy WS. A weight initialization method for improving training speed in feedforward neural network. *Neurocomputing* 30, 1-4 (2000), 219–232.

[144] Yang, Zitong et al. Rethinking bias-variance trade-off for generalization of neural networks. *arXiv preprint arXiv:2002.11328* (2020).

[145] Yeom, Samuel et al. Privacy risk in machine learning: Analyzing the connection to overfitting. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE. 2018, 268–282.

[146] You, Haoran et al. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957* (2019).

[147] Zhou, Hattie et al. Deconstructing lottery tickets: Zeros, signs, and the supermask. In: *NeurIPS*. 2019.

[148] Zhu, Chen et al. Transferable Clean-Label Poisoning Attacks on Deep Neural Nets. In: *ICML*. 2019.

[149] Zhu, Ziyun and Dumitras, Tudor. FeatureSmith: Automatically Engineering Features for Malware Detection by Mining the Security Literature. In: *CCS*. 2016.