Università Ca'Foscari Venezia

Master's Degree programme
in Computer Science

Final Thesis

AMEBA: An Adaptive Approach to the Black-Box Evasion of Machine Learning Models

Supervisor
Ch. Prof. Stefano Calzavara

Assistant supervisor
Ch. Prof. Claudio Lucchese

Graduand
Lorenzo Cazzaro
Matricolation number
864683

Academic Year
2020/2021

Abstract

Machine learning models are vulnerable to evasion attacks, where the attacker adds almost imperceptible perturbation to a correctly classified instance so as to induce misclassification. In the black-box setting where the attacker has no knowledge of the target model and only has limited query access to it, traditional attack strategies exploit the transferability property, i.e., the empirical observation that evasion attacks often generalize across different models. The attacker can thus rely on the following two-step attack strategy: (i) query the target model to learn how to train a surrogate model approximating it; and (ii) craft evasion attacks against the surrogate model and submit them to the target model, hoping that they "transfer". Since the two phases are assumed to be strictly separated, this strategy under-approximates the possible actions that a real attacker might take and it doesn't allow the attacker to exploit at the best the limited budget of queries to maximize the number of successful evasion attacks.
In this thesis we present AMEBA, the first adaptive approach to the black-box evasion of machine learning models. In particular, AMEBA is built on a well-known optimization problem, known as Multi-Armed Bandit (MAB). We describe the reduction from the two-steps evasion problem to the MAB problem, that allows one to exploit a MAB solving algorithm, the Thompson sampling algorithm, to define the new attack strategy. As a result, AMEBA infers the best alternation of actions for surrogate model training and evasion attack crafting. We choose both binary and image classification datasets to test AMEBA. Moreover, the ML models involved in the experiments vary between simple linear models, non-differentiable models and convolutional neural networks. Our experiments show that AMEBA outperforms the traditional two-steps attack strategy and is perfectly appropriate for practical usage.

Keywords Adversarial ML, Evasion Attacks, Machine Learning, Security, Supervised Learning

Contents

Chapter 1

Introduction

Machine Learning (ML) has become phenomenally popular in recent years and it has found a wide range of practical applications, many of which security sensitive like in self-driving cars, intrusion detection systems and object detectors. The more critical is the task for which the ML model is used, the more the ML model has to be robust and reliable. Yet it is now acknowledged that the adoption of ML in security-oriented applications should be done with care [6]. Indeed an attacker may confound the attacked model for her malicious objectives. For this reason, the adversarial machine learning field have gained more and more attention in the last ten years. The literature is rich of studies about the security of *supervised learning algorithms* and, in particular, their applications to *classification* tasks, where ML models are trained to predict one out of a set of possible classes, e.g., spam vs. ham.

A prominent class of threats against ML is constituted by the *evasion attacks*, that targets the integrity of the attacked model [6]. In an evasion attack, the attacker starts from an instance which is classified correctly by a ML model and perturbs it so as to induce a misclassification [7]. Indeed a slightly and intelligent manipulation of a correctly classified instance, different from the standard noise of data, may fool the machine learning classifier. There are plenty of examples: the attacker may corrupt the image of a panda through tiny pixel perturbations, which are imperceptible to humans, yet suffice to fool a ML model and to induce it to predict a gibbon with high confidence [45, 20]; the attacker may misspell words likely to appear in spam emails and add good words to make a spam email predicted as legit; finally, a client may modify carefully some personal information to obtain insurance coverage even if he doesn't possess the requirements to obtain it.

Depending on the information available to the attacker, evasion attacks are *white-box* or *black-box* [6]. White-box attacks assume the attacker to know everything about the model under attack, e.g., the learning algorithm, the complete feature representation of data, the training set and the model hyper-parameters. Black-box attacks, instead, only suppose a partial knowledge of the representation of the information by the attacker and additional constraints on accessing the target model make the attack difficult. For example, the attacker may have only query access to the model under attack (i.e., she can only ask for predictions). Black-box attacks are particularly important from a practical perspective, since this minimal capability is inherent to the model functionality.

A traditional approach to the black-box generation of evasion attacks exploits a subtle and surprising property known as *transferability*, i.e., the empirical observation that evasion attacks often generalize across different models [34]. A practical impact of this property is that it makes possible black-box attacks based on the use of a surrogate model whose details are completely known by the attacker. Then evasion attacks are crafted against the surrogate model and submitted to target model to verify if they transfer, i.e. they evade also the target model, or not. Hence, the attacker can adopt the following two-step attack strategy:

1. *Surrogate Model Training*: the attacker queries the target model to extract information about its behavior and builds a surrogate model approximating the target;

2. *Evasion Attack Crafting*: the attacker crafts successful evasion attacks against the surrogate model and feeds them to the target model, hoping that they "transfer" to it, i.e., lead to misclassification by the target.

This approach is appealing, because the attacker can use a surrogate model such that crafting successful evasion attacks against it is feasible using known algorithms for white-box attacks. For example, evasion attack crafting algorithms like the Fast Gradient Sign Method (FGSM) [20] work for any differentiable model. Though some prominent ML models are not differentiable, e.g., decision trees, the attacker can train a differentiable surrogate model, attack it through FGSM and then evade a non-differentiable target model via transferability.

1.1 Contributions

In this work, we question the effectiveness of the two-step attack strategy proposed in previous work [34] and briefly reviewed above. In particular, we observe that there is a *tension* between the two steps of the attack strategy. On the one hand, the attacker needs to query the target model multiple times in order to disclose its behavior and train a faithful surrogate model. On the other hand, the attacker wants to query the target model with as many evasion attacks as possible to maximize the number of misclassifications. This means that when the number of queries to the target model is limited, e.g., because the attacker pays a price for each query or wants to behave surreptitiously, the optimal attack strategy is far from straightforward.
Here we propose to move away from the two-step attack strategy of previous work and we present a new *adaptive* attack strategy, which dynamically learns whether queries to the target model should be leveraged for surrogate model training (step 1) or for evasion attack crafting (step 2), thus making the two steps of the attack intertwined. Our proposal subsumes the traditional two-step attack strategy of prior work, making black-box evasion attacks more effective and practical by automatically dealing with the delicate tension briefly described above.
To sum up, the contributions of this work are:

1. We propose the first adaptive approach to the black-box generation of evasion attacks against ML models. Our technique builds on a connection between the black-box evasion problem and a traditional optimization problem, known as Multi-Armed Bandit (MAB) [40]. In particular, we show how the black-box evasion problem can be reduced to MAB, hence it can be solved using existing approaches like the Thompson sampling algorithm [39]. We call the resulting attack strategy AMEBA (Adversarial Multi-armEd BAndit).

2. We implement AMEBA and we show it at work on different datasets, considering multiple ML models and attackers. We experimentally show that, at worst, AMEBA accurately approximates the behavior of the optimal two-step attack strategy, where the attacker leverages an oracle to find the best moment to switch from step 1 to step 2. At best, instead, AMEBA leads to the creation of a large number of evasion attacks which cannot be crafted even by the optimal two-step attack strategy. This shows that future research on adversarial ML should take adaptive attack strategies into consideration.

The AMEBA attack strategy and almost all the experimental results discussed in this thesis constitutes the work presented in [10] of which the author of this thesis is co-author. This work was presented by the author of this thesis at the ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS) 2021 on 8th June 2021[1].

1.2 Thesis Structure

The presented work is organized as follows. In Chapter 2 we introduce the necessary background related to supervised learning, the different machine learning models used in this work, adversarial machine learning, evasion attacks and the MAB problem. In Chapter 3 we provide a general overview about the most important categories of black-box attacks. In the first part we detail the black-box setting and we describe some types of black-box attacks, while in the second part we examine the black-box evasion problem and we provide a more detailed description of the two-step attack strategy through a prominent example from the literature. Following it in Chapter 4, we present our contribution, AMEBA, an adaptive attack strategy. In particular, we describe the threat model considered, the reduction from the two-step evasion attack problem to the MAB problem and the implementation of AMEBA. In Chapter 5, the experimental results are presented, among with the settings considered and the methodology adopted. Finally, in Chapter 6 the thesis is concluded with a brief summary of the work, then we provide a critical comparison with important work in the literature, in order to highlight our contribution, and finally we describe a list of open issues and research directions.

[1]https://dl.acm.org/doi/10.1145/3433210.3453114

Chapter 2

Background

In this chapter we introduce the concepts and the methods necessary to appreciate the rest of the work. The chapter is organized as follows. In Section 2.1 we discuss general concepts about ML and classification. Then in Section 2.2 we define the supervised learning algorithms and classifiers used in the experimental evaluation to test AMEBA. In Section 2.3 we provide the definition of Adversarial Machine Learning, the attack taxonomy and the definition of evasion attack along with two examples. Finally, in Section 2.4 we present the Multi Armed Bandit problem, a specific Reinforcement learning problem, and we describe one of its solving algorithms.

2.1 Machine learning

Machine learning is the field of computer science in which are studied algorithms that are able to learn from data [19]. The activity of learning follows the inductive method: derive laws and rules by using particular observations to make predictions about the future or unseen data. Every learning problem is classified by the type of feedback available to algorithm involved: the two most representative cases are unsupervised and supervised learning. The problem of unsupervised learning involves learning useful properties of the input data when no specific output values are supplied [38], for example learning the entire probability distribution that generated a dataset. On the other side, the supervised learning problem consists in learning a function from examples of its inputs and outputs [38], such as image and labels in image recognition problem. Since all this work deals with the classification task, the class of supervised learning problem will be deepened more formally.

2.1.1 Supervised Learning

Let $\{(\vec{x_i}, y_i)\} \in \mathcal{X} \times \mathcal{Y}$ be a set of couples built from the instances $\vec{x_i}$ of the set of instances $\mathcal{X}$, entities of the domain of interest, and y_i values from the set of labels Y and associated to the instances. The mapping between the instances in $\mathcal{X}$ and the labels in $\mathcal{Y}$ is defined by the unknown function $f : \mathcal{X} \to \mathcal{Y}$. $\mathcal{Y}$ is very often a set of strings or a set of real numbers, while the elements of $\mathcal{X}$ can be represented in many different ways that determine the nature of the prediction task. In this work it's supposed that the instances are represented through the vector model, so as a sequence of features, very often numerical. More precisely, the combination of d features is represented as a d-dimensional column vector, called feature vector, defined in the d-dimensional space ($\mathbb{R}^d$ if the features are real numbers), then an instance $\vec{x}$ is represented as $\begin{pmatrix} \vec{x}^{(1)} \\ \vec{x}^{(2)} \\ \vdots \\ \vec{x}^{(d)} \end{pmatrix}$.

The inductive inference task associated to the supervised learning problem consists in returning a function $\hat{h}$ that approximates f by learning from a training set $\mathcal{D} = \{(\vec{x_i}, y_i)\}_{i=1}^{n} \subseteq \mathcal{X} \times \mathcal{Y}$ [38]. The hypothesis h belongs to the hypothesis space $\mathcal{H}$, the set of hypothesis functions considered. The function $\hat{h}$ is returned by the learning algorithm $L : (X \times Y)^n \mapsto \mathcal{H}$ that maps a set of instances to the function $\hat{h}$ in $\mathcal{H}$ such that $\hat{h}(\vec{x}) \approx y$ in a predictive way. The function $\hat{h}$ that better approximate f can be found by following the empirical risk minimization [48]. Given a non-negative real valued loss function $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ that measures the cost of the mispredictions by the function $h \in \mathcal{H}$ with respect to the ground truth, $\hat{h}$ is the function that minimizes the empirical error:

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, h(\vec{x_i}))$$

, where $(\vec{x_i}, y_i) \in \mathcal{D}$.

An important property that has to be granted by the function learned $\hat{h}$ is generalization, i.e., it becomes possible to use the empirical error on $\mathcal{D}$ by $\hat{h}$ as a mirror of the expected error of $\hat{h}$ on the entire set $\mathcal{X} \times \mathcal{Y}$, in order to detect whether the function chosen is implicitly poor for the task. The choice of the hypotesis space $\mathcal{H}$ has a fundamental role in guaranteeing generalization. For example, suppose that the prediction task consists in finding a separating boundary of two classes of instances in a d dimensional vector space. If the predictive function $\hat{h}$ has the form of a linear separator and data are not linearly separable, it could not be the best assumption since it's not possible to determine a boundary that split the space in regions in which only instances of a certain class reside. More complex hypothesis imply more complex predictive functions $\hat{h}$ learned, that will be more tied to data from which the functions are learned, implying more variability as the training set $\mathcal{D}$ changes. In practice, it is common to allow complex predictive functions if the amount of available training data is sufficiently large, while it's more convenient to use less complex predictive functions with few data available.

2.1.2 Classification

If Y is the a finite set of labels, then the function f performs classification. The learning task becomes the classification task, that consists in finding the function $\hat{h}$, called classifier, that best approximates f and allows to classify new instances. The actual function used for classification can be directly $\hat{h}$ or, in other cases, $k : \mathcal{X} \rightarrow \mathcal{Y}$, that has the form $k(x) = g(\hat{h}(\vec{x}))$, where $h : \mathcal{X} \mapsto \mathbb{R}$ is the principal function learned that returns a value from $\vec{x}$ that, given to $g : \mathbb{R} \mapsto \mathcal{Y}$, allows to return the label of $x \in \mathcal{X}$. The classifier $\hat{h}$ partitions the space into class-labeled decision regions.

2.1.3 Performance Evaluation

There exist different performance metrics to evaluate a classifier $\hat{h}$ in the task of predicting the labels of a set of instances $\mathcal{S} \subset (\mathcal{X} \times \mathcal{Y})$, drawn from the same distribution of the instances in $\mathcal{D}$. Let $\mathcal{S}_t \subset \mathcal{S}$ the set of instances correctly classified by $\hat{h}$, the accuracy of $\hat{h}$ on $\mathcal{S}$ is $\frac{|\mathcal{S}_t|}{|\mathcal{S}|}$ [48]. The accuracy can be used independently of the nature of the classification task, binary or multiclass.
A key aspect to consider is the nature of the set S. It's not suggested to compute a performance metric on the same training-set $\mathcal{D}$ on which the function $\hat{h}$ was learned, since it's not possible to see how well the model generalizes on new data if new data aren't used. Then part of data available has to found a test set $\mathcal{T} \subset (\mathcal{X} \times \mathcal{Y})$ such that $\mathcal{T} \cap \mathcal{D} = \emptyset$: the performance of the learned function $\hat{h}$ on $\mathcal{T} = \mathcal{S}$ it is a reasonable estimate of the performance on new data.
However, reserving part of the data available as test set could not be optimal, since we don't exploit all the information available for training and, then, we risk to underestimate the performance of the classifier on new data. An overall summary of the performance of h on new data can be obtained by using the cross validation technique [48]. It consists in dividing the training set $\mathcal{D}$ into k subsets, typically 5 or 10 apply the learning algorithm L on $k-1$ subsets and test the predictive function obtained on the held-out subset. The score of cross validation is the mean of the result of the performance evaluation metric selected over all k tests. The fact that the learning algorithm and the correspondent functions are generated and evaluated on subsets of the training sets allows to avoid overfitting and to reduce the variability in the estimation of the expected performance of $\hat{h}$.

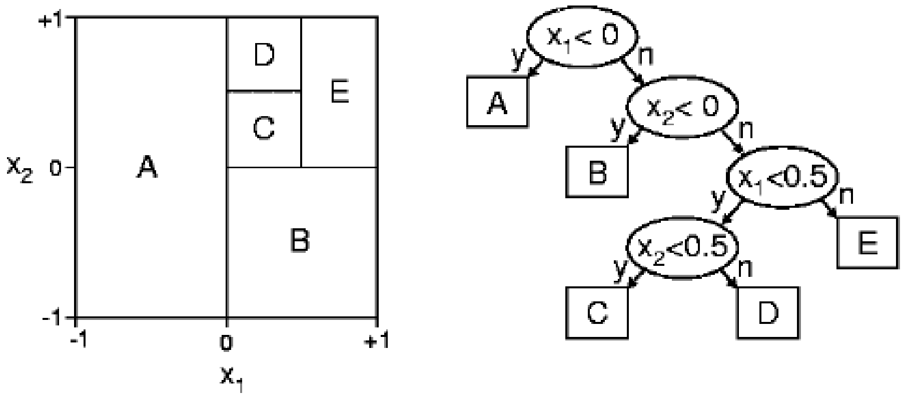


Figure 2.1: Decision tree and induced space regions, credits [36]

2.2 Classifiers

A great number of classifiers were developed since the born of AI. Nowadays, although neural networks are widely adopted, each classifier continues to be used at least in specific application domains. As consequence, they continue to be studied from the point of view of robustness and security. This section introduces classifiers that will be used in the experiments.

2.2.1 Decision Tree

A **decision tree** is a classification model that works by partitioning the input space into regions, whose edges are aligned with the axes of the feature space, and then assigns a prediction (for example, a label) to each region [8]. Given an instance $\vec{x} \in \mathcal{X}$, the process of selecting the prediction can be described by a sequential decision making process corresponding to the traversal of a tree [8].
Each internal node of the tree is annotated by a test on a feature (e.g. $\vec{x}^{(1)} < 1$ or $x^1 \in \{green, blue\}$, where $\vec{x}^{(1)}$ is an attribute of the instance $\vec{x}$), each branch represents the test result and each leaf bears the label of a class. The label assigned to a leaf will be the prediction for every instance whose features values allow to reach the leaf by following the outcomes of the tests. The thresholds used by the different internal nodes induces a partitioning of the feature space, so a leaf of the tree groups the instances of a region into a single class. This concept is represented in Figure 2.1.
The instance $\vec{x}$ is classified by starting from the root of the tree and following a path down to a specific leaf node, according to the outcomes of the tests of the internal nodes on the value of the features of $\vec{x}$.

Training

The classification trees are built using a top-down approach and a recursive, greedy and divide-and-conquer algorithm [21]. The approach is top-down since it begins from the root of the tree and the nodes are subsequently inserted. The algorithms is greedy because the node inserted is labeled with the test representing the best split of the partition of the training set considered at that particular step of the building process. Initially, the entire training set $\mathcal{D}$ is examined at the root of the tree and, if the instances don't belong to the same class, the splitting criterion is chosen. This criterion indicates which attribute and test determine the best way

to partition the set of instances into partitions $\mathcal{D}_j$. The quality of the splitting is measured by a attribute selection measure, that indicates the pureness of the splitting, i.e., if the sets of the partition contain only instances belonging to the same class. The splitting criterion is chosen such that the resulting partitions are as pure as possible, while the number of partitions created depends on both the type of features used and the attribute selection measure. Then the algorithm repeat the same process recursively to build the other branches and nodes from the partitions, that are examined in the subsequent nodes, and removes the attribute used in the previous splitting. In particular, if the attribute is numerical, the algorithm removes the attribute from the candidates used in the following nodes, otherwise it removes all the categorical values of the feature that was considered in the splitting. The algorithm stops only if either all the instances of a partition belong to a single class or the aren't remaining features to select for a stopping criterion or a stopping criterion is satisfied, like a maximum number of leaves of the tree (max leaf nodes) is reached. Then a leaf is added and labeled with the majority class in the considered partition of the training set.
The key parameter of the learning algorithm is the attribute selection measure. It's used for ranking in descending order the features that can be used to split the partition of the training set considered at a given node and selecting the best. Two criterion are used in this work: Entropy and Gini index. The Entropy criterion tends to prefer unbalanced splits in which one partition is smaller than the others [21], while the Gini index criterion enforces only binary splittings at each node [21].

2.2.2 Forest of Decision Trees

An **ensemble model** is a model that combines n models (or base classifiers) $M_1, M_2, ..., M_n$ to make up a $M*$ model [21]. The predictions given by $M*$ are dependent on the individual predictions of the classifiers that compose it. The training set $\mathcal{D}$ is used to create n training sets $\mathcal{D}_1, \ldots, \mathcal{D}_n$, where $\mathcal{D}_i,\ 1 \leq i \leq n$ is used to train the classifier M_i, that's called base learner. Ensemble models are extremely popular since combining some weak learners in order to create one strong is easier than designing a very strong one.
There are different ensemble models based on decision trees. Their aim is to make possible the use of decision trees in such way that the variance and the bias of the resulting ensemble model are lower than the ones of a single decision tree. The base learners must be the most different from each other, but also the more accurate as possible.
The method used for building the ensemble characterizes the ensemble model itself. The same learning algorithm for decision trees is used for every tree learned in the ensemble and the ensemble method influences the accuracy of the base learners and the independence between them. Since in reality it's very difficult to achieve the true independence between the base learners, the introduction of randomness in the learning algorithm helps.

Random Forest

One of the most popular ensemble model based on trees is Random Forest. It exploits the *bagging* algorithm [48] to learn the ensemble classifier. It works as follows: from the training set $\mathcal{D}$ are built n training sets $\mathcal{D}_i$ by bootstrapping

(sampling with repetition) the samples from $\mathcal{D}$. Then each decision tree is built from each training set. This method allows to train the single decision trees on slightly different datasets, improving the independence between the predictors.
Moreover, Random Forest introduces a difference about the decision tree learning algorithm explain in 2.2.1: only a subset of the features is considered for selecting the splitting criterion at each internal node. This subset of features is sampled randomly and with the same probability distribution at each repetition of the building algorithm. Normally, if d is the number of features of the instances, $\log(d)+1$ or $\sqrt{d}$ features are sampled for each node construction. This random part helps to obtain the independence between decision trees and better generalization since a very important predictor will not surely be chosen as part of the splitting criterion at the root of each tree [26].
The aggregation of the different base learners is obtained using the majority voting criteria: the prediction of the ensemble model is the most predicted label by the individual models. Random Forest is not prone to overfitting even if the number of trees used is large, so a high number of trees is often used in order to obtain a significant gain in accuracy [26]. Moreover, the training algorithm can be parallelized, since each subtree is trained over a different and independent training set.

AdaBoost

Another extremely popular ensemble method is AdaBoost, based on the *boosting* [48] ensemble method. The method starts by assigning weights to the different tuples of the training set $\mathcal{D}$. After the training of the i-th classifier, the weights of the tuples are updated so that the next classifier M_{i+1} can be trained more likely on the tuples wrongly classified by M_i. The prediction of the overall model thus obtained is the result of the combination of the weighted votes of each classifier that composes it. The weight of each classifier in the prediction is a function of its accuracy on the training set $\mathcal{D}_i$ on which it was trained [21].
In AdaBoost, each training instance has initially a weight $\frac{1}{n}$, where $n = |\mathcal{D}|$. The generation of k base classifiers requires k rounds of the algorithm. At round i, the dataset $\mathcal{D}_i$ is sampled with replacement by using the weights of the instances as sampling probabilities. Then the error of the classifier M_i is computed on $\mathcal{D}_i$ and if the classifier is too poor, it is discarded. The weights of the each instance in $\mathcal{D}$ is incremented if is incorrectly classified by M_i, it's descreased if correctly classified and remains equal if the instance considered wasn't in $\mathcal{D}_i$.
Boosting allows to reduce the model bias, as sub-models specialize in classifying instances incorrectly classified by the previous model, during the training phase. Normally a little number of short trees are used, since if each tree is too deep or too many of them are used, the model can overfit easily the training data. Compared to bagging, boosting tends to achieve higher accuracy [26].

2.2.3 Logistic Regression for Binary Classification Tasks

The logistic regression model [26, 27] is a generalized linear model that computes the probability that an instance $\vec{x} \in \mathcal{X}$ belongs to a particular category $y \in \mathcal{Y}$ by using the linear combination of the values of the features of $\vec{x}$. Let's assume that $\mathcal{X}$ has dimensionality d and the features of $\vec{x}$ can be modeled by a finite vector

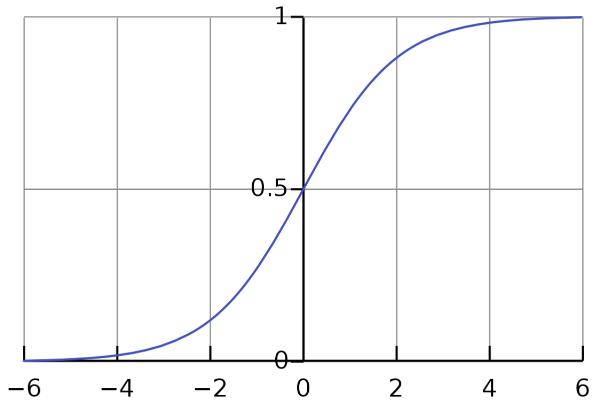


Figure 2.2: Logistic function, credits [1]

of real random variables X of size $d+1$ (it contains 1 as first element). Given the vector of real coefficients β of size $d+1$ learned by the training set, the linear combination of the features values is $z = X^T\beta$. Since z ranges over all $\mathbb{R}$, it's necessary to map the real value on the range $[0,1]$ to express the probability of the 1 outcome by using the logistic function $\hat{y} = \sigma(x) = \frac{e^x}{1+e^x}$, $x \in \mathbb{R}$, whose plot is in Figure 2.2, one of the functions that gives values between 0 and 1 [26]. Then the logistic model is expressed as follows:

$$p(X) = \frac{e^{X^T\beta}}{1+e^{X^T\beta}}$$

Training

The coefficients in β are estimated using the *conditional maximum likelihood estimation.* The objective is to maximize the log probability of the true y_i labels in the training data given the observations $\vec{x}_i$ [27]. Typically, the parameters estimated $\hat{\beta}$ solve the following optimization problem:

$$\hat{\beta} = \arg\max_{\beta} \sum_{i=1}^{n} (y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)) - \alpha R(\beta)$$

[27].
The objective is to find β that maximizes the log-likelihood function or, more commonly, that minimizes the opposite of the log-likelihood function, that becomes the objective *loss function.* The loss function ranges from 0, in the case in which all the instances with label 1 the predicted probability is 1 and 0 for the others, from infinity, in the opposite case. The global minimum of the loss function can be found by using *gradient descent* method and exploiting the convexity of the loss function, that guarantees that there exists only one minimum of the function which is also global.
In order to avoid overfitting, the regularization term $R(\beta)$ is added to the log-likelihood function. This term has the effect of shrinking the values of the parameters in $\hat{\beta}$ [27], thus reducing the variability of the estimate $\hat{\beta}$ and regularizing the model. The higher α, the higher the importance of the regularization term. For example, if the value of α tends to infinity, all the parameters will tend to a value near 0.

In this work the regularization term used is the L_2 *regularization*, that uses the square of the 2-norm of the weight values, e.g. $R(\beta) = \sum_{i=0}^{d} \beta_i^2$. Machine learning frameworks that offers logistic regression as model, such as *scikit-learn* [35], allows to set the regularization of the model through the parameter $C = \frac{1}{\alpha}$, then the higher C, the smaller the regularization of the model.

2.2.4 Linear SVM

Linear Support Vector Machine [8, 26] is a discriminative classifier that learns the decision boundary that separates the instances $\vec{x} \in \mathcal{X}$ in the training that belong to two classes with labels $y \in \{-1, 1\}$. The separating hyperplane learned has equation $< w, x > +b$, where w is the d dimensional vector of weights, b the intercept and d is the dimension of the features space $\mathcal{X}$.
In order to assign one class to a instance, Linear SVM uses the sign of the signed distance of the instance from the separating hyperplane. More formally, given the instance $\vec{x} \in \mathcal{X}$ and the classes $y \in \{-1, 1\}$, the Linear SVM classification function is defined as

$$\hat{h}(\vec{x}) = \text{sign}(w^T \vec{x} + b)$$

, where w and b are learned through the learning algorithm on the training set $\mathcal{D}$.

Training

In order to undestrand the optimization problem solved to find the parameters w and b of the best separating hyperplane, it's necessary to discuss SVM from the geometric point of view. The Linear SVM training algorithm finds the separating hyperplane that maximize the geometric margin, the distance of the hyperplane to the closest training points. Given an instance $\vec{x_i}$, the label y_i and the hyperplane $< w, x > + b$, the geometric margin is proportional to the perprendicular distance of the point $\vec{x_i}$ from the hyperplane. It's defined as

$$\gamma_{\text{geom},\vec{x_i}} = \frac{(w\vec{x_i} + b)y_i}{||w||}$$

[8]. For the closest points of the two classes to the hyperplane, called support vectors, the two parameters w and β are chosen such that $(w\vec{x_i}+b)y_i = 1$. Then the geometric margin for the support vectors becomes $\gamma_{\text{geom},\vec{x_i}} = \frac{1}{||w||}$, while $(w\vec{x_i} + b)y_i \geq 1$ for any other point. The euclidean distance between the closest points to the hyperplane selected is $\frac{2}{||w||}$. In order to maximize the geometrical distance from the closest points to the hyperplane and then maximize the margin, it's necessary to maximize the quantity above or, in other terms, minimize $||w||^2$.
Since data may be non linearly separable and the constraints on the margin may make the hyperplane too sensible to individual observations, the constraints on the margins are relaxed such that the hyperplane does not perfectly separate the two classes and some training instance may violate the margin or also the hyperplane side.
The optimization problem solved is then formulated as:

$$\text{minimize } \frac{1}{2}||w||^2 + C \sum_i \zeta_i$$

$$\text{subject to } y_i(w \cdot \vec{x_i} + b) \geq 1 - \zeta_i \wedge \zeta_i \geq 0$$

[8].
Intuitively the margin is maximized, but every instance that's inside the margin or at the wrong side of the hyperplane induces a penalty. Indeed for every $\vec{x} \in \mathcal{D}$ should held $y_i(w \cdot \vec{x_i} + b) \geq 1$ but this is not possible if the dataset is not linearly separable. For this reason, a slack variable ζ_i is associated to every instance $\vec{x}_i$ and it has value greater than 0 if $\vec{x}_i$ isn't at the right side of the margin. C is the regularization factor that controls how strictly the margins must be enforced. The bigger C, the hardest the margin is imposed, the less errors in classifying the training instances can be done and the narrower are the margins [8]. A low value of C allows to find a separating hyperplane that produces more errors in classifying points of the training set but the model generalizes data better.

Relation to Logistic Regression

Although the idea of finding the best hyperplane that separates the data of the two classes, allowing also some violations, seemed innovative when SVM was introduced, deep connections between SVM and other statistical methods have emerged [26]. One of those is the connection with logistic regression.
Indeed both the objective functions of the minimization problems solved in order to train the logistic regression and linear SVM model can be rewritten in the form

$$\underset{\beta}{\text{minimize}}\{L(\mathcal{D}, \beta) + \lambda P(\beta)\}$$

, where $L(\mathcal{D}, \beta)$ is a loss function quantifying the quality of fitting the data $\mathcal{D}$ while the term $P(\beta)$ is the penalty function on the parameters of the model controlled by the non negative constant λ. For linear SVM, $\beta = (w, b)$. From this, it emerges that the loss functions of the two models are very closely related [26]. While the value of the loss function used by the learning algorithm of Linear SVM is 0 for instances that reside at the correct side of the margin, the loss function of the logistic regression problem is non-zero everywhere but near to 0 for instances classified with very high probability of belonging to one or the other class, i.e. they are far from the decision boundary. Due to this similarities, the two models performs similarly [26]. However, SVM is preferred when the two classes are well separated, otherwise logistic regression is preferred [26].

2.2.5 Feedforward Neural Networks

A deep feedforward neural network [19, 2] is a ML model that consists in an hierarchical composition of n different functions of the input $\vec{x}$ that depend on some parameters learned in the training phase. Each function in the hierarchy represents a layer of the neural network and the overall length of the chain is the depth of the model. The final layer of the network is the output layer. Since the neural network should approximate, through training, a function $f(\vec{x})$, the output layer should return a label $\hat{y}$ for the instance $\vec{x}$ such that $\hat{y} = f(\vec{x})$. The other layers have not a specified behaviour and, for this reason, are called hidden layers. The model is associated with a directed acyclic graph describing how the functions are composed together [19].

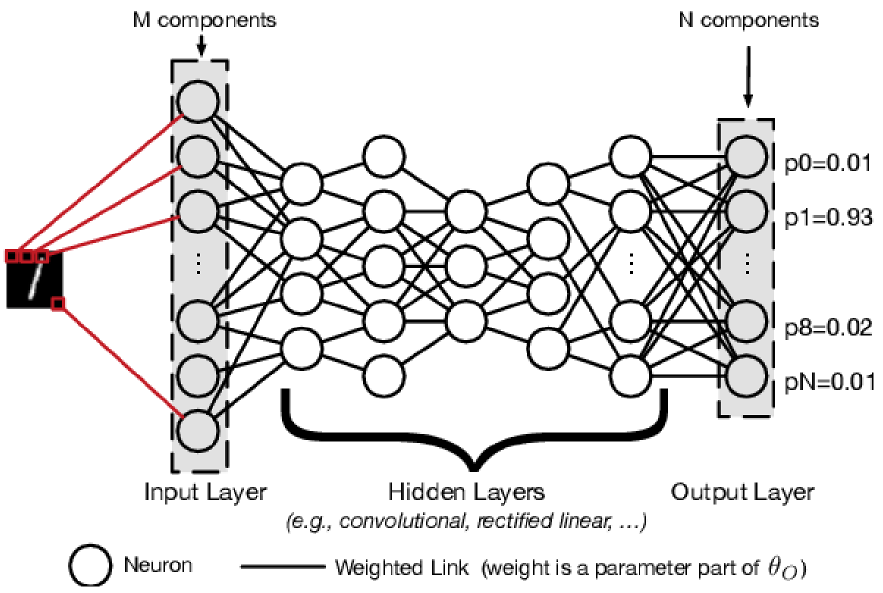

M components
N components
p0=0.01
p1=0.93
p8=0.02
pN=0.01
Input Layer
Hidden Layers
(e.g., convolutional, rectified linear, ...)
Output Layer
Neuron
Weighted Link (weight is a parameter part of θ_O)

Figure 2.3: Deepforward neural network graph representation. Credits from [34]

The graph representation allows to explain the model as composition of layers of nodes, called also neurons. The input layer contains d nodes, as the number of features of the instance $\vec{x} \in \mathcal{X}$, that don't perform any computation. Each node i of the input layer passes the value of the feature $\vec{x}^{(i)}$ to the node j of the subsequent layer through the weighted edge with weight w_{ij}. Each neuron of the second layer applies an activation function $\Phi : \mathbb{R} \to \mathbb{R}$ to the sum of the weighted features from the input layer, with the optional addition of a bias term b, incorporated as the weight of an edge by using a bias neuron with activation value 1. The returned value is z_i:

$$z_i = \Phi\left(\sum_{i=1}^{d} w_{ij}\vec{x}_i + b\right)$$

Then each neuron of the first hidden layer provides the value computed to each neuron of the second hidden layer that will apply the activation function (the same of the previous layer or another one) to the weighted sum of the incoming inputs and so on, until the output layer is reached. In Figure 2.3 an example a graph representation of a deepforward neural network is presented, which takes as input an image and return a vector of proabilities, one for each class.

The dimensionality of the layer j is the number p_j of the units in that layer and the column vector representation of the outputs of layer j is indicated as $\bar{h}_j$, with dimensionality p_j. The weights of the connections between the input layer and the first hidden layer are contained in a matrix W_1 with size $d \times p_1$, whereas the weights between the rth hidden layer and the $(r+1)$th hidden layer are denoted by the $p_r \times p_{r+1}$ matrix $W_(r+1)$. If the output layer contains o nodes, then the final matrix W_{k+1} is of size $p_k \times o$. The d-dimensional input vector x is transformed into the outputs using the following recursive equations:

$$\bar{h}_1 = \Phi(\vec{x}^T W_1)$$
$$\bar{h}_{p+1} = \Phi(\bar{h}_p^T W_{p+1}) \;\; \forall p \in \{1 \dots k-1\}$$
$$\bar{o} = \Phi(\bar{h}_k^T W_{k+1})$$

[2].
The activation functions are applied in element-wise fashion to their vector arguments and are supposed to be the same for each node in a layer. An extremely popular activation function is Rectified Linear Unit (ReLU), defined as $\Phi(v) = \max\{v, 0\}$.
The choice and number of output nodes is also tied to the activation function, which in turn depends on the application at hand [2]. In $k-$way classification, k output nodes with a softmax activation function may be used as output layer, defined as the following for the ith output z_i:

$$g(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \;\; \forall i \in \{1, \dots, o\}$$

The outputs corresponds to the probabilities of the k classes associated with x. Moreover, the choice of the activation function is critical and the most popular activation function nowadays is the ReLU function, defined as $g(v) = \max\{0, v\}$, applied by all the neurons of a layer.
The description of the neural network as a composition of layers of neurons justifies the formalization of the model as an optimized composition of functions. Indeed, supposing that at layer r the units compute the functions $g_1, \dots, g_{p_r}$, while the function f is applied by every node of the layer $r+1$ to their input. Then the node at the layer $r+1$ computes $f(g_1, \dots, g_{p_r})$. All the functions computed in each layer depend on the weights of the connection between neurons learned during the learning phase, that optimize the weights jointly.

Training with the Backpropagation Algorithm

The standard training algorithm for deepforward neural network is the backpropagation algorithm [2], that allows to learn the weights of the model such that the chosen loss function L is minimized.
The weights w_{ij} of each layer are typically initialized to random values. Then, at every iteration of the algorithm, the weights have to be updated in order to minimize the loss function L on the training data. Let $\hat{y}_0, \dots, \hat{y}_{C-1}$ the probabilites of the $C-1$ classes predicted for an instance $\vec{x}$ that belongs to one the classes. The loss function typically adopted in the multiclass classification setting is the cross-entropy loss, defined as $L = -\sum_{i=0}^{C-1} y_i \log(\hat{y}_i)$, where $y_i = 1$ if $i = y$, the label associated with $\vec{x}$, otherwise $y_i = 0$. Since $\hat{y}_i$ predicted by the NN is provided by a composition function applied to $\vec{x}$, also L becomes a complicated composition of functions that depends on the weights learned. The backpropagation algorithm allows to update the weights w_{ij} leveraging the multivariate chain rule of differential calculus that express the error gradients as summations of local-gradient products over the various paths from a node to the output.
The backpropagation algorithm contains two main phases, the forward and backward phases. An epoch of training is concluded when the backpropagation algorithm is executed over the entire training set. The number of epochs is established

before the training phase or by using a stopping criteria during the training phase, such as the convergence of the weights. The two phases of the algorithm are the following:

- Forward phase: the instance $\vec{x}$ is passed to the neural network, the input neurons are feed and the features values are used to compute the post activation value for every node in each hidden layer based on the current weights. The value of the output o is also computed.
- Backward phase: the main goal of the backward phase is to update the weights w_{ij} using the gradient of the loss function with respect to the different weights. This phase is backward since the weights are updated in the backward direction, starting from the output layer.

  In particular, the weights are updated using gradient descent method, then $w_{ij} = w_{ij} - \eta \frac{\partial L}{\partial w_{ij}}$. An important parameter of the method is the learning rate η, with value typically between 0 and 1, that multiplies the update component A good value of the learning rate is essential in order to find a global minimum solution and not local.

In this work, mini-batch stochastic gradient descent is used. Firstly the training set is divided into random batches of instances, then the weight updates are compute over the batch of instances and finally the weights are updated at the end of the batch iteration. This method allows firstly to not have all training data in memory but only the batch required, reducing memory costs, and allows for a faster convergence, avoiding local minimum [2]. However, the batch size is an additional hyperparameter that must be tuned. The typical batch size are at the power of 2, typically 32, 64, 128 or 256 instances.

2.2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [2, 19] are designed to work with grid-structured inputs, which have strong spatial dependencies in local regions of the grid [2]. The most obvious example of grid-structured data is a 2-dimensional image. Since in this work CNNs are used in a image classification task, CNNs description will be contextualized to image classification. This architecture was proposed for the first time by LeCun et al. [30].
Figure 2.4 shows an example of CNN architecture. In CNNs, each layer of the network is 3-dimensional grid structure, which has a width, height and depth. The depth represents the number of features maps in the hidden layer and the number of color channels in the input layer. Three types of layers are normally present in a CNN: convolutional, pooling and ReLU. This sequence of layers is devoted to extract features from the input. Moreover, a final set of layers is often fully connected, or dense, in order to use the features extracted from the previous part of the network to provide a prediction through the output nodes.

Convolution

A convolutional layer consists in a set of 3D grids composed of neurons like the feedforward neural network, whose connections to the previous layer are sparse but carefully designed [2]. The parameters, weights of the connections, are arranged

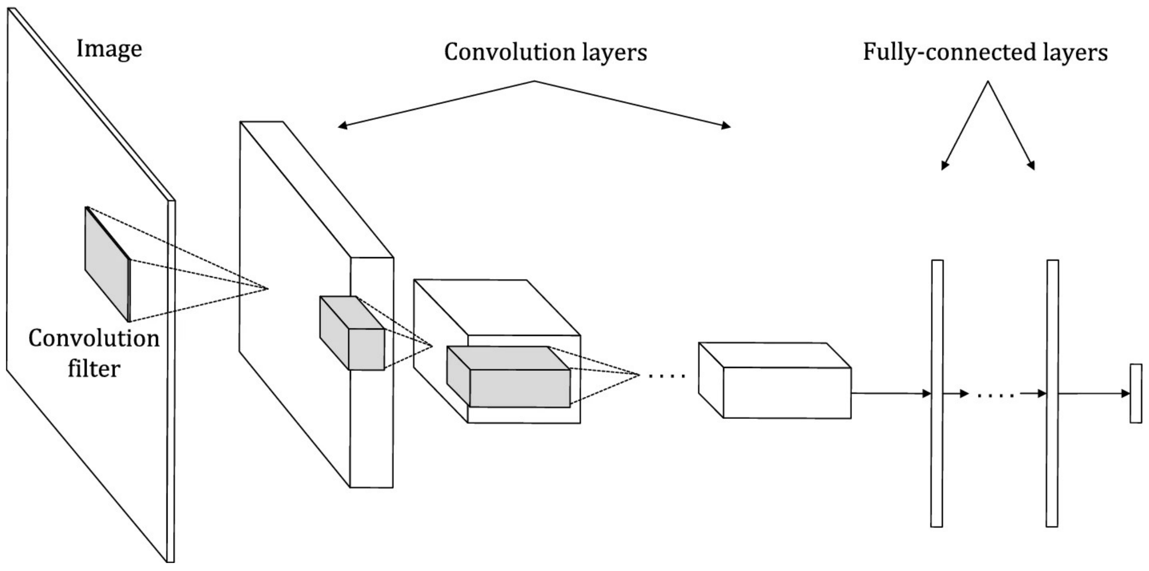


Figure 2.4: Example of architecture of a CNN with convolutional layers and final fully connected layers. Credit [22]

into sets of 3D structural units called filters or kernels. A filter is usually a square in terms of width and height, smaller than the dimensions of the input layer, while its depth matches the one of the layer on which its applied. The filter is involved in the convolution operation that defines the dimensionality of the next layer of the network. Figure 2.5 shows an example of the convolution where the depth of the filter is 1. Convolution imposes the filter at each possible position of the input layer of neurons so that the filter fully overlaps with the layer. Each position in which the filter can be placed defines a node with input value that's the dot product between the weigths of the filter and the region of the input layer covered. Since the filter is repeatedly shifted along all the layer in order to compute the feature map of the subsequent layer, the parameters of the filter remains the same across all the convolution process. From the point of view of the nodes in the grid structure, the same filter parameter is used as the weight of the connection between the feature nodes in the subsequent layer and the nodes in the actual layer that appear at the same position in the filter. The depth of the layer is established by the number of the filters used in a certain layer.

The convolution operation is parametrized by a stride and a padding parameter. These two parameters are used to control the size of the resulting feature maps with respect to the size of the kernel and the input layer. The stride defines how many steps we are moving in each step of the convolution. For example, stride 2 implies that the filter is moved two positions at every step, resulting in a smaller feature map produced. The padding parameter represent the number of columns/rows of neurons that return input 0 that will be added symetrically at the borders of each feature map of the input layer. These nodes don't contribute to the input value of the nodes to which are connected. Padding allows a filter to stick out from the borders of a layer and increase the dimensionality of the resulting feature map.

The number of filters used in each layer control the number of weights of the CNN, since the each parameter of the filters is trainable. Different layers can use different number of filters. The idea about the usage of filters is that each filter tries to identify a certain type of spatial pattern inside a certain region of the input mage, such as an horizontal edge, and a large number of filters is necessary to capture a

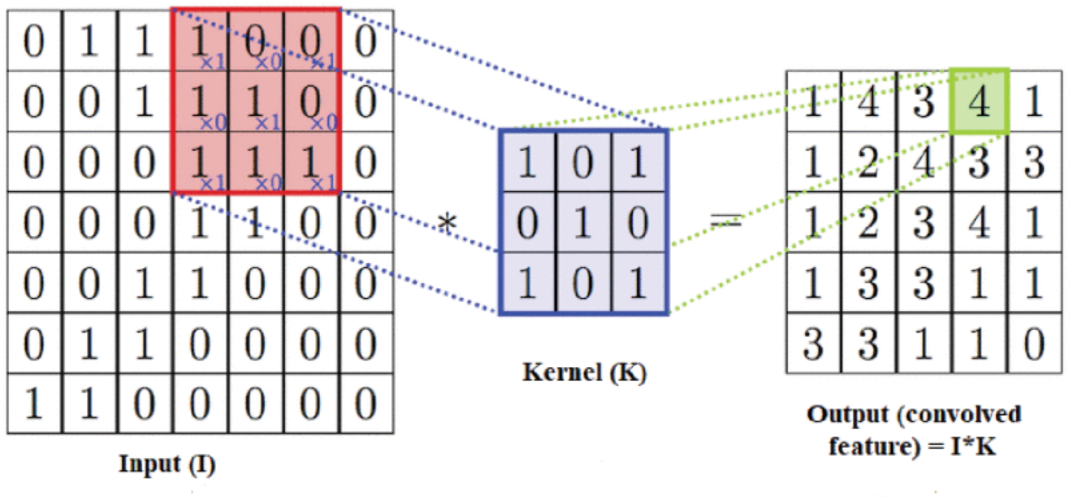


Figure 2.5: Convolution. Credits [42]

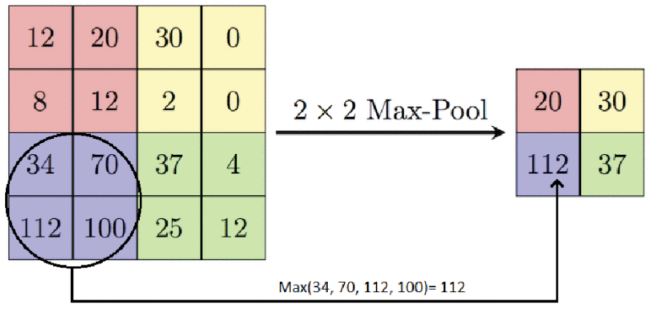


Figure 2.6: Maxpooling. Credits [42]

great variety of the possible patterns combinations. Each node, whose weights are defined by a filter, is said to capture a feature through the filter. The filters in the early layers are used to capture more primitive spatial shapes, while the filters in the later layers tend to have smaller area but a wider depth, in order to capture complex features [2].

Typically, at each layer, it is applied the ReLU activation function to the output value of each node in the feature map after the computation of the weighted sum of its input values.

Pooling

The pooling operation is applied to regions of each feature map of a layer and produces another layer with the same depth as the layer on which is applied. It consists in compacting the values of regions of each feature map by applying a function to retrieve the maximum value (max pooling) or mean value (average pooling) in that region. Pooling drastically reduces the spatial dimensions of each activation map, above all if also the stride is used. In practice, max-pooling is extremely popular and it's very often interleaved with convolutional/ReLu layers. An example of how max-pooling works is in Figure 2.6.

Early Stopping

Training a neural network by optimizing the loss function up to convergence may induce the overfitting of the training data, i.e., the neural network doesn't generalize to test data since it becomes excessively adherent to the training data, learning also the inherent noise.
Early stopping [19] the training procedure allows to mitigate this problem. A part of the training data is used as validation data and backpropagation is applied on the training data. Then the training procedure is stopped when parameters updates no longer cause an improvement of the NN on the validation set. The parameters of the NN are set to the value reached in the epoch in which the network reached the best validation error.

Data Augmentation

NN typically requires an huge training set in order to achieve the desired performance and not overfit the set provided. Data augmentation [19] allows to make up for this requirement by increasing the number and diversity of instances in the training set. It consists in generating new instances from the dataset by applying such transformations on the original instances that allows to maintan the same class. In the context of image processing, transformations like rotation, translation, zooming and reflection are used, since they do not fundamentally change the properties of the object on the image. However the choice of the operations and their paramaters should be done with care. For example the symmetric transformation with respect to the horizontal axis on a image containing a 6 results in a image containing the digit 9 and completely change the class of the image.

Dropout

Dropout is a regularization technique for NN. It was introduced by Srivastava et. al. [41] and it's useful to prevent overfitting.
Dropout is typically used with a minibatch learning algorithm. Each time an instance is submitted to the NN, each neuron of the NN or of a specific layer (except the output layer) is dropped from the NN along with its connections with probability p (normally $p = 0.5$). Then the forward and backward pass of the backtracking algorithm are executed without considering the dropped nodes, i.e. each involved layer of the NN has a different configuration. At test time, in order to provide predictions, the weights going out of each unity are scaled by the probability of sampling that unit.
Dropout present two important advantages. Firstly, dropout trains intuitively an ensemble of subnetwork that can be obtained by removing nonoutput units from the NN [19]. However dropout is efficient, since it doesn't require to train subnetworks separately and allows to provide a prediction with only a forward pass of the input instance. Secondly, dropout introduces some noise in the training process so that the units in a layer don't fix up the mistakes of the previous layer, situation also called layer co-adaption. This is one cause of overfitting, since the co-adaption don't generalize to unseen data [41]. Since under dropout the node of a layer are randomly sampled, the network result reduced in the number of parameters during training. For this reason, dropout should be used with wide networks.

2.3 Adversarial Machine Learning

Adversarial machine learning is the research field of computer science that deals with the vulnerabilities and the robustness of machine learning algorithms in the training and classification phase, under different threat models.

Even though the number of papers on adversarial machine learning have seen an exponential growth in the last years, the first seminar works in the field, such as Dalvi et al. [16], dates back to 2004. The few works at that time were in the context of spam filtering and they showed that linear classifiers could be easily fooled by manipulating carefully the emails without compromising their readability. In the meantime, Barreno et al. [3, 4] proposed an overview of the vulnerabilities of machine learning from a more general perspective, claiming for the need of adversarial machine learning, i.e., to develop learning algorithm robust against the attacks [6]. However, this field obtained considerable attention in 2014 after the publication of the work by Szegedy et al. [45]. Their work showed that deep neural networks misclassify adversarial examples, i.e., instances generated from correct classified samples using very small perturbations, such that the perturbed instances are nearly indistinguishable from the original ones. From this point, the work published in this field started increasing exponentially year on year and involved also the evaluation of other models, such as SVM, classifier that was also attacked previously in [17], and forests of decision trees, for example in [12].

The evaluation of vulnerabilities of a machine learning based system by using a proactive approach [6], i.e., before being attacked, is essential to anticipate the attacker. In order to design an attack against supervised learning-based system, is often exploited a popular framework based on the attack taxonomy [4, 23]. In particular, it's provided a taxonomy about the possible attacker's goals and knowledge. Moreover, the optimal attack strategy is defined as an optimization problem, whose solution provides the way in which perturb data in order to achieve the attacker's goal [6].

2.3.1 Attacker's Goal

The description of the attacker's goal completely determines the target of the attack. Among all the aspects, the attacker's goal is defined in terms of the specific security aspect targeted [6]. In particular, an integrity violation aims to evade the target model without compromise the machine learning system, while an availability violation aims to compromise the normal functionalities of the system. In case of integrity attack, the error specificity diffentiates further the objective [6]. In a target attack, the attacker aims to evade a system and obtain a specific (targeted) prediction, while a generic (untargeted) attack aims to only evade the target system.

2.3.2 Attacker's Knowledge

The attacker may have different level of knowledge of the attack system/supervised learning algorithm used. Depending on this, three attack scenarios are devised by Biggio et al. [6], different in the attacker's knowledge and difficulties as detailed below.

White-box Setting

The attacker knows everything about the target system, then she knows the target model, its parameters, the dataset on which it was trained and the features of the instances considered by the model. This setting is the ideal for the worst-case evaluation of security of learning algorithms.

Gray-box Setting

This scenario is based on the attacker's limited knowledge of the components well known in white-box attacks. Indeed the attacker is supposed to know, typically, only the feature representation and in many cases the type of learning algorithm. However, even if the attacker knows the kind of classifier to attack, it could be too computationally expensive to craft the attack sample against it. In this scenario the attacker is assumed to be able to gather a surrogate dataset from a similar source as the target model training set, since it clearly knows the feature representation. Then the surrogate dataset may be used for training a surrogate model, that enables the optimization of the attacks against it instead of forging an attack directly against the target classifier.

Black-box Setting

On the other side, in the black-box setting the attacker doesn't know the attacked model, its parameters, the training set used and the representation of the instances adopted. However, in order to make possible perform an attack, the attacker has to know at least the task performed by the classifier attacked, which kind of features are used by the model and the potential transformations to apply to cause features changes. All this information is often inferred by reasoning: if a system performs spam classification, it is very likely that it will exploit a bag of words representation of the email in input, that's the frequency of a vocabulary of terms inside the email. The attacker also knows the nature of data used to train the attacked model, even if in this case she may not know the exact representation. In this scenario, the attacker has to retrieve information by submitting queries to the classifier attacked. The feedback received can be a label or the probabilities of belonging to a certain class. The attacker may also exploit the generation of the attack based on the surrogate classifier, even if the features used to describe the instances may be different from the ones used by the target model.
The black-box setting is particularly important from a practical prospective, because of the minimal capabilities of the attacker.

2.3.3 Evasion Attacks

Evasion attacks consist in manipulating input samples in order to cause misclassification by the attacked classifier. By definition, these attacks occur at test time. For example, the attacker may want to manipulate carefully an image to fool an object detection system or she may want to modify a malware in order to have it recognized as a legitimate program [6]. Evasion attacks aim only to violate the integrity of the attacked classifier, since the functionalities of the attacked system are not compromised.
More formally, let $\mathcal{X}$ the feature vector space and $\mathcal{Y}$ a finite set of class labels. The classifier $h : \mathcal{X} \to \mathcal{Y}$ is a function that assigns a label to every point $\vec{x} \in \mathcal{X}$.

The classifier h tries to emulate the human understanding of the set of instances to classify, represented as the function $f : \mathcal{X} \to \mathcal{Y}$ that assign the correct true label (ground truth) to every instance. Suppose that the attacker can manipulate an instance $\vec{x} \in \mathcal{X}$ following a set of feasible perturbations and the resulting instance $\vec{z} \in \mathcal{X}$ is classified with the same label of the original instance by f. We denote the set of perturbed instances as $\Phi(\vec{x})$ such that $\forall z \in \Phi(\vec{x}) : f(\vec{x}) = f(\vec{z})$. $\Phi(\vec{x})$ contains all the instances that preserves the same label as $\vec{x}$ and that respect specific application bounds on the features values.
Then, given the classifier h and an instance $x \in \mathcal{X}$ such that $h(\vec{x}) = f(\vec{x})$, an evasion attack against the classifier h is any instance $z \in \Phi(\vec{x})$ such that $h(\vec{x}) \neq h(\vec{z})$.
In this work we use error generic or untargeted evasion attack, evasion attack that aims to have $\vec{z}$ classified as belonging to any class except y, the class to which $\vec{x}$ belongs to.

Error Generic Evasion Attack

Even if the attacker knows all about the attacked system, a concern is how perturb an instance in a sensible way. Indeed, for the most of the attack strategy, is infeasible in practice to enumerate all the possible instances perturbed in $\Phi(\vec{x})$ and an attacker may want to not perturb an instance in a way without a precise reason. The most popular approach is to formulate the error generic evasion attack as a maximization problem of a loss or objective function that incorporates information about the target model. The formal definition of error-generic evasion attack is given by Biggio et al. [17].
Let consider a loss function ℓ associated with the target classifier h. The objective of the attacker is to maximize that function to pursue her goal, i.e., craft an instance $\vec{z} \in \Phi(\vec{x})$ that evade h, also called *adversarial example*. The optimization problem to solve in order to perform the evasion attack against h is:

$$\begin{aligned} \max_{\vec{z}} \ & \ell(\vec{z}, y) \\ \text{s.t. } & ||\vec{z} - \vec{x}||_p \leq \epsilon \\ & x_{lb} \leq \vec{z} \leq x_{ub} \end{aligned} \tag{2.1}$$

where $||\mathbf{v}||_p$ is the ℓ_p norm of $\mathbf{v}$, ϵ a real value that constraints the norm value, x_{lb} and x_{ub} the lower and upper limit value of the features of the perturbed instance $\vec{z}$ and $\epsilon \in \mathbb{R}^+$. The objective of the attacker is to find the instance $\vec{z} \in \Phi(\vec{x})$ such that it maximizes the loss function with respect to the original label y so it will be hopefully misclassified. The loss function ℓ may coincide with the function that describes the loss incurred by h in classifying $\vec{z}$ as y. However, the loss function of the target classifier may be not known, for example in gray box or black box setting, then ℓ has to be designed by the attacker. In particular, the loss function ℓ is defined depending on the knowledge of the attacker and influence the formulation of the optimization problem.

2.3.4 Adversarial Examples Crafting

Adversarial examples are defined as examples (instances) that are slightly different from correctly classified instances drawn from the data distribution [20]. They have the property to be very similar to instances correctly classified by the target model,

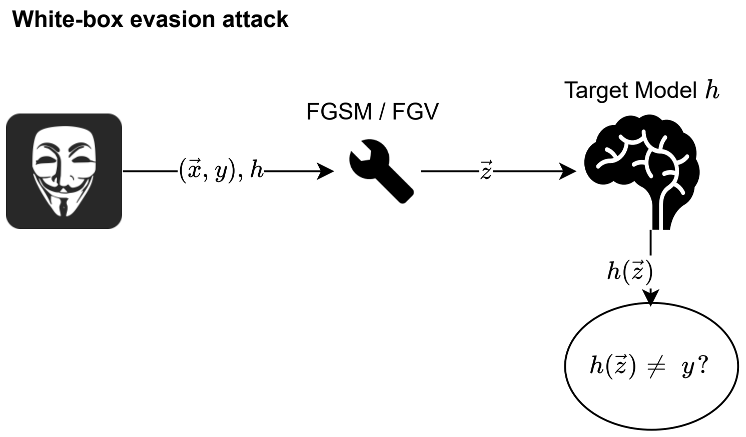


Figure 2.7: Adversarial example crafting

but they are misclassified. The generation of adversarial example is the result of the application of an evasion attack against a machine learning model. In practice, it consists in solving the optimization problem stated in section 2.3.3, where the loss function ℓ has to be carefully designed. In Figure 2.7 is described the general workflow for crafting an adversarial example against a known target model.
Among the different strategies for crafting adversarial examples, one of the most popular, simple and fast white-box alternatives is Fast Gradient Sign Method (FGSM) [20] proposed by Goodfellow et al. against neural networks. In this work, a slightly different attack methods is used, derived from FGS, called Fast Gradient Value [37], to compute adversarial examples against a CNN and Linear SVM.

Fast Gradient Sign

The Fast Gradient Sign (FGS) method [20] was one of the first attacks for crafting adversarial examples against neural networks. It becomes very popular for its simplicity, efficiency and efficacy, though now are available white-box evasion attacks more reliable, such as Carlini and Wagner attack (C&W) [11]. Since neural networks are mostly exploited in computer vision, this attack was designed for generating adversarial examples from images using neural networks.
Let's suppose the attacker wants to generate an adversarial example $\vec{z}$ from the instance $\vec{x} \in \mathcal{X}$ classified as $y \in \mathcal{Y}$. Moreover, suppose that the constraint on the perturbation uses the ∞ norm, then $||\vec{z} - \vec{x}||_\infty \leq \epsilon$. Since this attack was presented against neural networks, the standard approach is to define the loss function ℓ optimized in the evasion attack optimization problem 2.1 as the loss function J optimized in the training phase of the neural network. The attack strategy provide the linearization of the loss function and define the optimal max-norm perturbation as:

$$\eta = \text{sign}(\nabla_{\vec{x}} J(\vec{x}, y))$$

Using η, the adversarial example z is generated as

$$z = \vec{x} + \epsilon\, \eta = \vec{x} + \epsilon\, \text{sign}(\nabla_{\vec{x}} J(\vec{x}, y))$$

The attack is said to be one-step gradient based, since the base instance x is perturbed only once. This method produces an adversarial example in which group of features are perturbed in the same way without taking into account the magnitude of the gradient along different components, producing a sparse attack.

In case of images, it produces an indistinguishable blurring effect if ϵ is accurately chosen.

Fast Gradient Value

Fast Gradient Value (FGV), an extension of the FGS attack, was proposed by Rozsa et al. [37] to fool neural networks. The idea is to consider the scaled version of the raw gradient of the loss function optimized instead of the sign of the gradient. Suppose that the distance constraint of the optimization problem 2.1 uses the ℓ_2 norm, i.e. $||\vec{z} - \vec{x}||_2 \leq \epsilon$. Then, if we linearize approximatively the function ℓ around $\vec{x}$, for the parameters θ of the target model considered, then the attempt to solve the optimization problem constitutes in generating the adversarial example with the perturbation:

$$\eta = \frac{\nabla_{\vec{x}}\ell(\vec{x}, y)}{||\nabla_{\vec{x}}\ell(\vec{x}, y)||}$$

, the adversarial example z is crafted by:

$$\vec{z} = \vec{x} + \epsilon\ \eta = \vec{x} + \epsilon\frac{\nabla_{\vec{x}}\ell(\vec{x}, y)}{||\nabla_{\vec{x}}\ell(\vec{x}, y)||}$$

. This is an ℓ_2-norm attack where the instance $\vec{x}$ is moved along the direction of the gradient of the loss function.
In this work, FGV is used to compute adversarial examples against CNNs and Linear SVM. In the original formulation of FGV, the target model is a neural network with loss function $J(\vec{x}, y)$, so the optimization problem considered becomes the problem 2.1 with ell_2 norm constraint. Then the attack becomes:

$$\vec{z} = \vec{x} + \epsilon\frac{\nabla_{\vec{x}}J(\vec{x}, y)}{||\nabla_{\vec{x}}J(\vec{x}, y)||}$$

However, the same principle can be used to attack also other models, changing the loss function considered. Against SVM, Papernot et al. [33] presented a simple attack against multiclass SVM that exploits the same principle of FGV. Since in this work a linear SVM for binary classification tasks is attacked, we consider the adapted version of the attack that's:

$$\vec{z} = \vec{x} - y\epsilon\frac{\nabla_{\vec{x}}(\mathbf{w}^T\vec{x} + b)}{||\nabla_x(\mathbf{w}^T\vec{x} + b)||} = \vec{x} - y\epsilon\frac{\mathbf{w}}{||\mathbf{w}||}$$

where $\mathbf{w}$ is the gradient of the discriminative function of the linear SVM, i.e. $\nabla_{\vec{x}}(\mathbf{w}^T\vec{x} + b) = \mathbf{w}$, also called the weight vector.
The advantage of the FGV resides in the fact that the use of the raw gradient allows to not ignore the differences in gradient magnitude between corresponding features of the instances as FGS does, then the loss function optimized is effectively increased with smaller perturbations [37]. Moreover FGV generates more focused perturbations, since all components are perturbed depending on the gradient magnitude for that component, while FGS method perturbs group of features depending on the sign of the corresponding components of the gradient. This could be a very important for some tasks like image classification, in which local perturbations of the image are preferred rather than perturbing uselessly also the background.

However, the formulations of the attacks above don't consider that the adversarial example $\vec{z}$ has to satisfy some constraints about the feature values. In order to take in account them, consider the clipping operator $\Pi_{\Phi(\vec{x})}$ that projects the instance $\vec{x}$ in the feasible domain $\Phi(\vec{x})$ where the features constraints are satisfied. Then the FGV attack for the two classifiers considered becomes:

$$\begin{aligned}\vec{z} =& \Pi_{\Phi}\left(\vec{x} + \epsilon \frac{\nabla_{\vec{x}} J(\vec{x}, y)}{||\nabla_{\vec{x}} J(\vec{x}, y)||}\right) \text{ against NN} \\ \vec{z} =& \Pi_{\Phi}\left(\vec{x} - y\epsilon \frac{\mathbf{w}}{||\mathbf{w}||}\right) \text{ against SVM}\end{aligned}$$

2.4 The Multi-Armed Bandit problem

2.4.1 Reinforcement Learning

Reinforcement learning is a machine learning paradigm in which an agent interact actively with an environment in order to achieve a goal despite the uncertrainty of the environment [43]. The environment is described by its state, that's sensed by the agent and it's influenced by the agent's actions. At each state, the agent has a set of possible actions available and she has to choose the most rewarding. A policy defines a mapping between a state of the environment and the action to be taken in that state. The goal of the agent is described in terms of the maximization of a reward over the long run. At each time step, the agent receives a reward signal by the environment in response to the action taken. The policy is influenced by the rewards received over time to select actions, for each state, that the environment state or the action selected to reach the goal. However, the value function specifies the total reward that the agent may expect to cumulate over the future, starting from a state [43]. Although the policy should guide the agent towards states of higher value, values have to be estimated from the sequence of actions and corresponding rewards observed, in order to allow to reach the goal of maximizing the total reward.

2.4.2 Definition of the MAB problem

Definition of the Problem

The Multi-Armed Bandit (MAB) is a well-known optimization problem that unifies different scenarios in which an algorithm makes decisions over time under uncertainty [40].
The problem setting is the following: let $K \geq 2$ the set of possible *actions* $\mathcal{A}$, also called *arms*, and $T \geq 1$ rounds, at every round one needs to choose an arm and collect a reward from this arm. In its most common formulation, known as MAB with *stochastic bandits*, the problem relies on three assumptions:

1. The reward is observed only for the selected action and nothing else. The rewards of the other actions that could have been selected at a a certain turn are unknown.

2. For each action $a \in \mathcal{A}$, there is a unknown distribution D_a over reals, called the *reward distribution*. Every time a is chosen, the reward r is indepen-

dently sampled from D_a. The distributions of the rewards of the arms are independent from each other.

3. Per-round rewards are bounded: the standard range for rewards is the continuous interval $[0, 1]$.

The reward distributions induce a *mean reward vector* $\mu \in [0,1]^K$, where $\mu(a) = \mathbb{E}[D_a]$ is the mean reward of the action a. The goal of a MAB solving algorithm is thus finding the sequence of actions $a_1, \dots, a_T \in \mathcal{A}$ which maximizes the *cumulative reward* $\sum_{i=1}^{T} \mu(a_i)$.
It's evident that any solving strategy has to deal with the *exploration-exploitation tradeoff.* Indeed it's necessary to *explore* the reward of the arms, in order to obtain a certain level of confidence on their estimates. On the other hand too many turns cannot be invested in exploring, since a proper number of turns has to be used for the *exploitation* steps, that consist in choosing the arms with the maximum expected reward, in order to maximize the cumulative reward.
A simpler variant of MAB with stochastic bandits assumes *Bernoulli bandits*, where any action yields either a success or a failure and the reward distributions are Bernoulli. In particular, the action $a \in \mathcal{A}$ produces a *success* and the reward 1 with probability θ_a, while a *failure* and the reward 0 is produced with probability $1-\theta_a$. Then the unknown mean reward vector μ thus coincides with the vector of the probabilities of success of each action, i.e., $\mu = (\theta_1, ..., \theta_K)$. A well-known solution to this variant of the problem is given by the *Thompson sampling* algorithm [39] that's a greedy algorithm.

2.4.3 Thompson Sampling Algorithm

The key idea of Thompson sampling is to sequentially learn the mean reward vector, during the *exploration* steps, and at the same time exploit the action with the highest mean reward in order to *exploit* the information obtained to maximize the total reward. The algorithm that's choosing the actions, called also the *agent*, has an independent prior belief over θ_a and this prior is beta distributed with parameters $\alpha \in (\alpha_1, \dots, \alpha_K)$ and $\beta \in (\beta_1, \dots, \beta_K)$. For each action a the prior probability density function of θ_a is

$$p(\theta_a) = \frac{\Gamma(\alpha_a + \beta_a)}{\Gamma(\alpha_a)\Gamma(\beta_a)} \theta_a^{\alpha_a - 1}(1-\theta_a)^{\beta_a - 1}$$

, where Γ is the gamma function, and the mean value is $\dfrac{\alpha_a}{\alpha_a + \beta_a}$. At each turn, the estimate $\hat{\theta_a}$ is sampled for each action $a \in A$ and the action with the maximum estimate is chosen. As the turns pass and the actions are taken, the distribution of the mean reward of each action is updated accordingly to the reward obtained. In particular, the posterior distribution of the mean reward r_{a_t} of the action a_t taken at round t is updated using a very simple rule:

$$\begin{cases} (\alpha_a, \beta_a) & \text{if } a_t \neq a \\ (\alpha_a, \beta_a) + (r_{a_t}, 1 - r_t) & \text{if } a_t = a \end{cases}$$

The the posterior distribution of the mean reward of the arm a becomes the prior distribution for the next round in which a is selected, since it will be updated

Algorithm 1 Thompson sampling

```
1: for $a$ in $\mathcal{A}$ do
2:     $(S_a, F_a) \leftarrow (1, 1)$                          ▷ Initialization
3: end for
4: for $t = 1, ..., T$ do
5:     for $a$ in $\mathcal{A}$ do
6:         Sample $\hat{\theta}_a \sim Beta(S_a, F_a)$
7:     end for
8:     $a_t \leftarrow \arg\max_{a \in \mathcal{A}} \hat{\theta}_a$
9:     $r_t \leftarrow$ PERFORM$(a_t)$                          ▷ Get reward of $a_t$
10:    $(S_{a_t}, F_{a_t}) \leftarrow (S_{a_t} + r_t, F_{a_t} + (1 - r_t))$
11: end for
```

by using the new reward. At the first round $\alpha_a = \beta_a = 1$ $\forall a \in A$, so $p(\theta_a)$ is uniform over $[0, 1]$ [39]. The parameters (α_a, β_a) are often called *pseudo-counts* and indicated as S_a and F_a, since whether the action a is successful then the former is increased by one, otherwise the latter.
Thompson sampling uses the posterior distributions, that are also the prior beliefs, to randomly sample the success probability estimates $\hat{\theta}_a$. Indeed the posterior distributions become more and more peaked around the effective estimated mean rewards as more data is seen, so the high variability of the posterior distribution of rarely took actions might lead to sample a value greater of the values sampled for other more pulled actions, allowing the exploration of less used actions. However, if the estimated mean reward of an action is greater than the others, in the average that actions will be more chosen than the others since its sampled value will be in the average greater than the others.
Algorithm 1 presents the pseudocode of Thompson sampling, where the PERFORM function takes the chosen action and returns the corresponding reward (0 or 1). At each round t, the action with the maximum sampled mean reward is chosen, the action is performed and the pseudocounts are updated accordingly.

Chapter 3

Black-box evasion attacks

The black-box attack scenario assumes the most limited knowledge of the target system by the attacker. Although the attacker cannot perform white-box attacks against the target model, machine learning can be threatened anyway. In this section we provide a complete description of the assumptions about the attacker's knowledge in the black-box setting and we provide a classification of the black-box attacks presented in literature. Moreover, we discuss the transfer-based two-step attack strategy.
The chapter is organized as follows. In Section 3.1 we briefly discuss the definition of attacker's knowledge in the black-box setting. In Section 3.2 we examine the different types of feedback/output provided by the target model in the black-box setting. Then in Section 3.3 we present in general different types of black-box attacks proposed in literature, in particular the transfer-based attacks. Finally, in Section 3.4 we provide the definition of the traditional two-steps attack strategy used to perform evasion attack in the black-box setting. We discuss its features, limits and we examine the first work that proposed it.

3.1 Attacker's Knowledge

We use the definition of attacker's knowledge in the black-box setting provided by Biggio et. al. [6].
In this setting, the attacker hasn't any substantial knowledge about the threatened ML model, the feature space used by the system, the learning algorithm and the training data [6]. However, the access to the target model and a minimal knowledge about the target system allow the attacker to carry out an attack anyway. Indeed the attacker knows the task for which the target classifier is designed and can exploit this knowledge to infer some useful information. Although she may not know the exact feature representation used by the target, at least she knows the kind of features used by the system and the potential perturbation that can be applied to an instance in order to change them. In the same way, she may at least know the kind of data used to train a classifier for a specific task, even though she doesn't know the exact training samples.

3.2 Feedback Available

In order to concretize the attack, the attacker must have the previlege to query the target model. However the feasibility of the attack depends also on the feedback

available to the attacker by the heterogenous target classifiers in the wild. Different scenarios could be defined depending on the attacker's query capabilities and the response received.
In the worst-case scenario, defined by Chen et. al. [13], the attacker acts in a no-box setting, where she is denied to query any information from the targeted classifier for adversarial attacks. This is the most challenging scenario, since it limits the assumptions on the black-box scenario further. It prevents all the attacks described below that requires either a minimum knowledge of the target model or a feedback for a given query. The only possible attacks feasible in the no-box setting are based on transferability.
On the other hand, the best-case scenario considers an unlimited query access to the target classifier [13]. The feedback received could consists in a wide range of information, for example confidence scores, classification probabilities for all the classes or ranking for classification. Due to the feasibility of attacking in this setting, although the attacker has not access to the details of the targeted model, this scenario has been called practical and is the typical scenario for prominent work in the area [11, 32, 33, 34].
However, Ilyas et. al. [24] proposed more realistic variant of the black-box settings. They include some limitations in the query access capabilities of the attacker and the feedback available that are typical of real-world systems. In particular, they distinguish three sub-scenarios:

- Query limited setting. In this setting, the attacker has a limited number of queries to the classifier [24]. Then the attack algorithm should generate an evasion attack in a efficient way, in terms of number of queries to the target. The limit can be due to (i) a limitation of resources, like a time limit for the inference time, (ii) a monetary limitation, if the attacker incurs in a cost for each query sent, (iii) a defense mechanism, like an intrusion detection system that limits the number of queries during an attack opportunity window. Examples of this scenario are the services offered by Clarifai NSFW[1], Google Cloud Vision API[2] and Amazon Machine Learning[3], three platforms that requires a per-query payment.

- Partial-information setting. In this setting, the attacker only has access to the probabilities $P(y|\vec{x})$, where $\vec{x}$ is an instance and y one of the top k classes predicted by the target classifier [24]. An online classifier that returns only the top k scores (not probabilities) is the one offered by Google Cloud Vision API or Clarifai NSFW.

- Label-only setting. In this setting, the adversary only has access to a list of k inferred labels ordered by their predicted probabilities. When $k = 1$, the attacker has only access to the top label predicted. As example, in [24] are reported photo tagging apps like Google Photo[4], that add labels to the user images without assigning a confidence or probabilities score to them.

The three scenarios are not mutually exclusive. Real-world systems offers services for automated classification than combine both the limitation in the number of

[1]https://www.clarifai.com/models/not-safe-for-work-image-recognition

[2]https://cloud.google.com/vision/pricing?hl=en

[3]https://aws.amazon.com/getting-started/projects/build-machine-learning-model/services-costs/

[4]https://photos.google.com/

queries and in the amount of information provided to the user through the prediction. Thereby the attacker has not only to find a strategy to make up for the lack of complete information about the target model used, but she should also take into account other access limitations.
Two types of evasion attacks have been receiving a lot of interest in the last years: the transferability-based and query-only attacks. Other strategies were also proposed, like boundary search attacks and hybrid approaches that exploit the best characteristics of the previous categories. In the next section these categories of black-box evasion attacks will be discussed also in relation to the attack scenarios described above.

3.3 Categories

3.3.1 Transfer-based

Transferability

Transferability-based evasion attacks are based on the transferability property of adversarial examples, that's the empirical observation that some adversarial examples produced to mislead a specific classifier $\hat{h}$ also can mislead other models h, even if the models greatly differs [33].
The transferability of adversarial examples crafted for a neural network to another neural network was initially assessed by Goodfellow et al. [19] and Szegedy et al. [45]. However, preliminary studies involved only neural networks, trained on different subsets of the same training set, without extending the analysis to other machine learning techniques.
Papernot et al. [33] presented the first study of transferability on a wide collection of ML models. In their work, they partitioned transferability into two variants and showed that both the variants affects ML models. The first is the intra-tecnique transferability, defined for models trained with the same machine learning tecnique but with different parametrizations or datasets (like two neural networks) [33], while the second is the cross-tecnique transferability, defined for models trained using different techniques [33] (like a Linear SVM and a forest of decision trees). Their experimental evaluation comprises a collection of representative ML models presented also in the background section 2.1 trained on the dataset of handwritten digits MNIST. Their experimental evaluation shows clearly that machine learning techniques are surprisingly vulnerable to both the types of transferability. However, the transferability property is not guaranteed for every adversarial example against every target model. As shown by Papernot et. al. [33] and Liu et. al [32], the transferability of adversarial examples may not reach 100% of the examples tested, but it can be low. In particular, Liu et. al. [32] distinguished between transferability of non-targeted adversarial examples that can assume any label except the original one, and target adversarial examples that have to be classified as belonging to a specific class different from the ground truth. They pointed out that although transferable non-targeted adversarial examples are easy to find, transferable targeted adversarial examples are more difficult to find.
As the transferability of adversarial examples was gaining popularity, an effort was made to understand deeply the property and its limits. Demontis et. al. [17] shown that the transferability deeply depends on three elements. Firstly, the complexity of the target model, its inherent vulnerability [17], make a ML model more or

less vulnerable. Then a more regularized model is more robust against evasion attacks based on transferability. Secondly, the transferability is impacted by the alignment between the two involved models, described as the alignment between the gradients of the respective loss functions. Intuitively, the more similar are the two models in terms of loss or prediction functions, the more effective is an attack based on transferability. Finally, a ML model with a stabler loss function, with lower variance, facilitates the generations of transferable adversarial examples. For this reason, it's better to use less complex and high regularized ML models as surrogate model.

The First Transfer-based Attack Proposal

Transfer-based attacks are particulary suitable in the black-box scenario, since the attacker doesn't need to know the details of the target model in order to perform the evasion.
The transfer-based evasion attack was firstly proposed by Biggio et al. [7], who were the first to treat the problem of optimal evasion at test time in a gray-box setting [6]. They provided the adaptation of their gradient descent attack [7] in their limited knowledge (LK) setting, that's more similar to the gray-box setting than the black-box setting (see section 2.3.2). However, they pointed out the main steps and some assumptions that a transferability-based evasion attack possesses:

- The attacker cannot directly evade the target $\hat{h}$;
- The attacker can collect a surrogate dataset $S' = \{(\vec{x_i}, y_i)\}_{i=1}^{n_q}$, with $n_q = |S'|$. They suppose that the samples are drawn from the same underlying distribution of the training set of the target model.
- The attacker use S' to train a surrogate classifier $\hat{h}$ that approximates the target h. In order to closely approximate h, the attacker should learn $\hat{h}$ using the labels provided by h.
- The attacker crafts an adversarial example $\vec{z}$ against $\hat{h}$ and uses it to evade h.

They used their evasion attack algorithm to attack a malicious PDF classifier in white-box and LK setting, where the objective is to modify PDFs containing malware such that they were recognized as legitimate, or negative, by the target classifier. In black-box setting, both surrogate and target classifiers are trained on two different partitions of the same dataset, but the size of the training set of the surrogate is only 20% of the cardinality of the target training set. Their experimental results showed that in the black-box setting, the false negative rate obtained attacking the target classifier using adversarial examples crafted against the surrogate classifier is slightly lower than the one obtained in the white-box setting, showing that only a small set of sample as surrogate training set may be required to successfully attack the target classifier [7]. They were the first to point out that transferability allows to realize evasion attack without a detailed knowledge of the target model. However their work presents two limitation: they don't analyze how to retrieve the surrogate training set in practice and they don't test the attack against a wide range of state-of-the-art classifiers to show that it works in a real scenario.

Two aspect were not deepened in this work: how to retrieve the surrogate training set and how to improve the transferability. They were object of subsequent work and nowadays they consitutes open lines of research.

Collecting the Surrogate Training Set

In the black-box setting, the attacker isn't supposed to know also the feature space $\mathcal{X}$ in which the instances classified by target h. However, the surrogate training set has to be collected.
The instances in the surrogate dataset haven't to be represented with the same features as the ones use in the training dataset of the target. However, the knowledge about the task performed by the target classifier and the input that it accept is very useful to infer the exact representation. For example, if the target classifier accepts only gray-scaled images with 28×28 pixels, then a convenient choice is to gather a dataset of digits of the appropriate dimension.
The surrogate training set can be obtained either using the domain knowledge of the classification task performed by the target [6] or by exploiting the query-access to the target model and labelling instances using its predictions [34], in order to improve the alignment between surrogate and target model. Moreover, it's also possible to use pre-trained models available online, especially in that case in which a very large dataset would be necessary to train the surrogate model. Examples are the pre-trained deep neural network architectures offered by machine and deep learning APIs, like Keras[5].
Retrieving a surrogate training set by asking the target model to predict the class of some instances is suitable both in partial information setting and the label only setting. In the partial information setting, is sufficient to consider as prediction the class with the gratest score as the predicted label. In case a large set of instances is not accessible, a small set of unlabeled instances can be used to build a synthetic dataset that uses the predictions of the target model on selected queries, as shown by Papernot. et. al. [34]. However, the limited number of queries could be a huge constraint in designing this type of attack. With a limited number of queries, the attacker should choose carefully how many queries will be spent in the two steps. Different strategies were proposed in order to reduce the number of queries needed for retrieving the training set of the surrogate model, for example by exploiting active learning strategies and selecting the most informative instances to be labelled [31].

Improving the Transferability

A different line of research focused on how to improve the transferability.
One of the factor that influence the transferability in a transfer-based evasion attack is the choice of the surrogate model. The results of the experiments performed in Demontis et. al. [17] and Papernot et. al. [33] shown that the choice of the surrogate model could have an important impact on transferability, but the phenomenon is partially unexplored and it's impossible to drawn general rules. As example, Papernot et. al. [33] shown that Deep Neural Networks results are more vulnerable to adversarial examples transferred from a Deep Neural Networks than from other models. However, SVM and decision trees don't allow to craft evasion attacks that transfer well against the same model even if the surrogate is trained

[5]https://keras.io/api/applications/

on a different training set of the target, even if the two models are trained on two subset of the same training set. Observations that emerges from both the two papers are:

- The use of a regularized model allows to obtain a better transferability.
- It's better to use a DNN as surrogate model against a target DNN.
- Every classifier can produce a satisfactory transferability against decision trees.

The knowledge of the task perfomed by the target classifier can give useful hints to choose an appropriate surrogate model.
Another attack method that improve the transferability was suggested by Liu et. al. [32] and Dong et. al. [18]. Instead of using a single surrogate model to generate an adversarial example, evasion attacks are crafted against an ensemble of models. The intuition is that if an adversarial example is adversarial for multiple models, then it may capture an intrinsic direction that always fools this models and is more likely to transfer to other models at the same time [32], enabling more powerful black-box attacks. Since evasion attack crafting is defined as an optimization problem 2.3.3, is necessary to define a loss function. In both the paper the loss function to optimize is defined as a loss function that combines the output functions of the Deep Neural Networks considered in the ensemble.

The Two-steps Attack Strategy

From the description of the transfer-based attack appears evident that the attack is articulated into two main steps: (i) the collection of a surrogate training set to train a surrogate model, (ii) crafting evasion attacks against the surrogate model, hoping that they transfer to the target model. The description of the strategy and how making it practical, i.e., applicable in a real-world scenario, was described for the time by Papernot et. al. [34]. Since this strategy consitute the building block of the transfer-based attacks and it's improved in this work, it will be deepened in section 3.4.

3.3.2 Query-based

The success rate of the black-box attacks based on transferability depends on dimension and quality of the training set used for the surrogate model, the surrogate model used and the white-box evasion attack used to generate adversarial examples. The limited success rate of the attacks led to the design of another type of black-box attacks more similar in spirit to white-box attack, i.e. without the need of a surrogate model. The query-only evasion attacks allow to compute an evasion attack against the target by approximating the gradient of an objective function defined on the gradient of the target model.
Evasion attacks based on zeroth-order optimization don't require the computation of a gradient directly on the model attacked. Firstly, they exploit the formulation of a loss or objective function for the problem at hand, that need to be optimized for crafting the adversarial example that depend on the output of the target classifier. Then, the optimization problem of the evasion attack (Section 2.3.3) is carried out by using classical algorithms like gradient descent with an estimated gradient,

obtained for example by using the symmetric difference quotient [13] that doesn't require the direct computation of the gradient, but uses two evaluations of the target function. More formally, given the objective function $g(\vec{x})$, the instance $\vec{x} \in \mathcal{X}$ and $x^{(i)}$ one of its components, the gradient $\frac{\partial g}{\partial x^{(i)}}$ (defined as $\hat{g}_i$) is computed as:

$$\hat{g}_i = \frac{\partial g(\vec{x})}{\partial x^{(i)}} = \frac{g(\vec{x} + he_i) - g(\vec{x} - he_i)}{2h}$$

where h is a small constant (as $h = 0.0001$ in [13]) and e_i is a canonical vector containing 1 only at the i-th component. However, the evaluation of such gradient could be impractical if the number of components is huge. As example, consider the estimation of the gradient of the loss function of a CNN that accepts in input a 299×299 pixels image. The number of components is $p = 89401$ and the total number of queries to the target CNN to estimate the gradient is $2p = 178802$, too much queries for a single gradient estimation. Indeed a single query to the target model may require tens of milliseconds or the machine learning system may impose some limitations on the maximum number of queries.

Then literature on this type of attacks proposes different techniques to reduce the number of queries needed to generate a successful adversarial example, evaluated with respect to the average time required to complete an attack and on the average number of queries to the target model needed. Chen et al. [13] proposed an iterative method mainly based on stochastic coordinate descent that choses at each iteration the most important component of the gradient such that along that direction the objective function is minimized (maximized) the most. However, the generation of a successful advexp on ImageNet[6] requires a huge number of queries against the Inception-V3 network [46]. Bhagoji et al. [5] proposed to group features and optimize the objective function by estimating the partial derivates along the directions determined by the groups of features, but their error generic single-step attack requires on average 200 queries per sample on the MNIST dataset and against a simple CNN. Illyas et al. [24] designed strategies based on zero order optimization in different feedbacks scanrios, listed in section 3.2, and they evaluated the strategies on ImageNet, showing that an advexp requires at least hundreds of queries. Cheng et al. [14] proposed a framework for attacking machine learning systems that return only the label of the query. This setting is described as really challenging since the objective functions used by previous work become discontinous and hard to optimize. However, they attack requires thousands of queries to reduce the refine the distortion applied to the adversarial example. Finally, Ilyas et al. [25] proposed to reduce the number of queries needed for the gradient estimation by using priors of the distribution of the gradient, but their approach requires on average more than one thousand queries to craft untargeted adversarial examples on ImageNet.

Though this line of research is promising because of the higher success rates of the attacks, similar to the ones obtained by white-box evasion attacks, the number of queries required by query-only attacks are still orders of magnitude higher than what can be achieved via surrogate model training.

Although transfer-based and query-only attacks constitutes the two major line of research, other attack strategy have been explored.

[6]http://image-net.org/

3.3.3 Hybrid Approaches

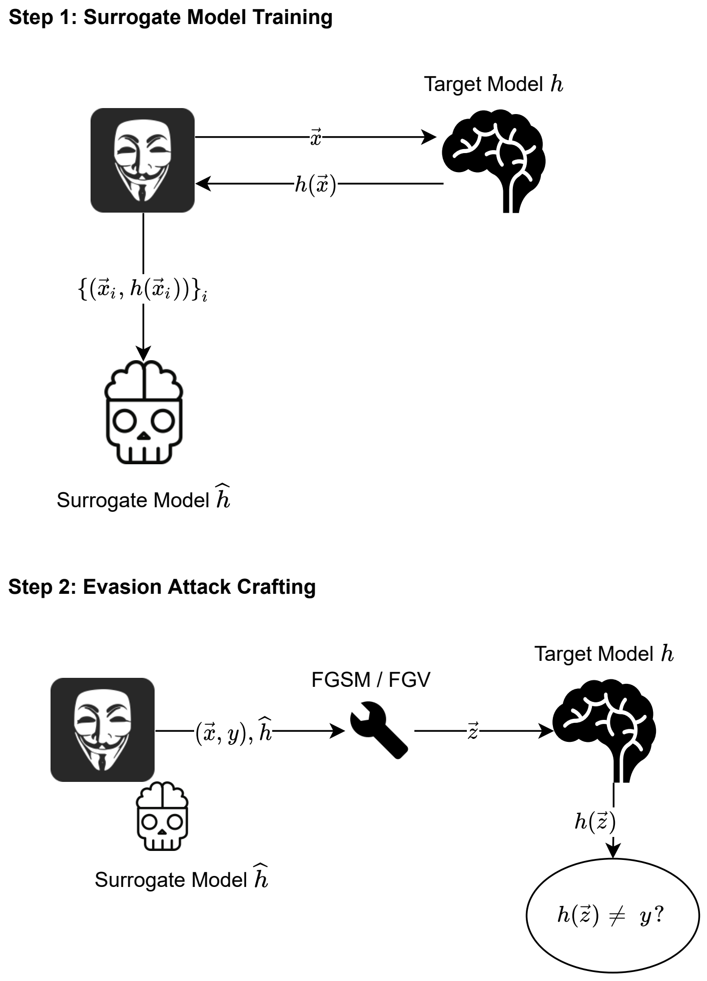
Attacks that try to combine the two main approaches presented above were proposed in literature. Cheng et al. [15] proposed a query-only attack that exploits the transfer-based prior for the estimation of the gradient of the target model. This prior is the gradient of a surrogate model trained on the predictions of the target model on carefully selected instances that helps to estimate the gradient of the target model. The results are surprising: on average, only 650 queries are necessary to forge an untargeted attack against Inception-V3 on ImageNet. Juuti et al. [28] proposed a targeted attack similar to the previous one, where an ensemble of models is used to estimate the gradient of the target model that returns only the top-k scores or probabilities for each a query. Their approach can be considered adaptive, since their complete attack consists in the combination of different attacks that are used depending on the number of queries needed for the evasion. For example, initially a tansfer-based attack is attempt and, if it's not effective, their new attack is used. Finally, Suya et al. [44] designed a query-efficient attack in which gradient estimation attack is used only for the non transferred adversarial examples, reducing the query cost when useful local models are available.

3.3.4 Decision Based Attacks

Another line of research deals with the boundary attacks, designed for the black box setting with minimum feedback from the target model (only the label of the instance). Brendel et. al. [9] proposed an attack based only on the prediction of the target model. In particular, the algorithm is initialized with an adversarial instance (classified incorrectly) and the base instance of the advexps. The adversarial instance is perturbed to perform a random walk along the boundary between the adversarial and the non-adversarial region. The objective is to maintain the adversarial instance in the adversarial region and reduce the distance to the base instance [9]. This attack is still competitive against gradient based attacks with respect to the minimal perturbation used to generate the adversarial example, with the advantage of being designed for the hardest black-box setting.

3.4 The Transfer-based Two-step Attack Strategy

As stated in the Background chapter, section 2.3.3, the goal of the attacker in an evasion attack is to evade the target classifier h at test time by means of adversarial examples. Many variations of the black-box setting were considered above. In one of the worst scenario, the attacker has only hard-label access to the target model, i.e. only the label of the query is returned, and the number of queries that the attacker can perform is limited.
The maximum number of exploitable queries is designed as the budget $B \in \mathbb{N}$ of the attacker. When the budget of the attacker is constrained to smaller values, like 3000 instances, the query-only attacks could not allow to generate a satisfactory number of evasion attacks, due to the average needed number of queries to generate even a single attack with very high efficacy. Then the attack based on transferability is more suitable, indeed the attacker may manually train or find a pretrained differentiable surrogate model $\hat{h}$ to generate attacks via transferability.



Step 1: Surrogate Model Training
Target Model h
$\vec{x}$
$h(\vec{x})$
$\{(\vec{x}_i, h(\vec{x}_i))\}_i$
Surrogate Model $\widehat{h}$
Step 2: Evasion Attack Crafting
Target Model h
FGSM / FGV
$(\vec{x}, y), \widehat{h}$
$\vec{z}$
$h(\vec{z})$
Surrogate Model $\widehat{h}$
$h(\vec{z}) \neq y?$

Figure 3.1: Two-step attack strategy

Even though there are a lot of pretrained models available, mostly very deep and memory expensive neural networks trained for perceptual tasks, it's more general to suppose that no pretrained model fit well the attack scenario.
As anticipated in section 3.3.1, the black-box evasion attack based on transferability is carried out following two steps:

1. *Surrogate Model Training*: the attacker queries the target model to extract information about its behavior and trains a surrogate model approximating the target;

2. *Evasion Attack Crafting*: the attacker crafts successful evasion attacks against the surrogate model and feeds them to the target model, hoping that they "transfer" to it, i.e., lead to misclassification by the target.

The two-step attack strategy is also described in Figure 3.1. This approach is appealing, because the attacker can train the surrogate model such that crafting successful evasion attacks against it is feasible using known algorithms. For example, evasion attack crafting algorithms like the Fast Gradient Sign Method (FGSM) [20] work for any differentiable model. Though some prominent ML models are not differentiable, e.g., decision trees, the attacker can train a differentiable surrogate model, attack it through FGSM and then evade a non-differentiable target model via transferability.

Algorithm 2 Substitute DNN training

```
Input: $\tilde{O}$, $max_\rho$, $S_0$, $\lambda$
1: Define architecture $F$
2: for $\rho \in\ 0 \dots max_\rho - 1$ do
3:     $D \leftarrow \{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\ \}$        ▷ Label the substitute training set
4:     $\theta_F \leftarrow \text{train}(F, D)$        ▷ Training F on D to evaluate parameters $\theta_F$
5:     $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$        ▷ Perform Jacobian-based dataset augmentation
6: end for
7: return $\theta_F$
```

The objective of the attacker is typically to obtain the highest number of successful evasion attack as possible and the highest transferability, given the small number of queries available.
In literature the two-step attack strategy was explored by studying different ways for minimizing the number of queries required in the training step, without studying how to dynamically interleave the two actions.

3.4.1 Practical Black-Box Attack

The first two-step approach for evading machine learning models using transferability was proposed by Papernot et. al. [34]. The strong point of their work consists in the two-step attack strategy in the label-only setting. They supposed that the attacker hasn't access to a large initial dataset. In this way, their attack is applicable also in the real world scenario, for example against a classifier hosted remotely by a third-party. They described both the general two-step attack strategy and their implementation that uses a heuristic to limit the number of queries used in the first step. We provide a brief description of their implemention in order to point out some criticalities of their work.

Description of the Particular Strategy Proposed

The chosen target model is a multiclass DNN classifier, called as the oracle O such that $\tilde{O}(\vec{x})$ is the label given to the instance $\vec{x}$. The attack strategy consists in training the surrogate model on a synthetic dataset generated by the attacker and labeled by the oracle. Then the adversarial examples are crafted against the surrogate model and transferred to the target model. The first objective is to show that the generation technique of the synthetic training set allows to train an approximation F of O. The second objective is to show that their attack is *pratical*, in the sense that requires a small amount of instances as an initial surrogate training set, since the rest of the dataset is synthetic.
Since the number of queries submitted to the oracle is limited in the wild, it's not possible to make an infinite number of queries to obtain the model output $\tilde{O}(\vec{x})$ for any input $\vec{x}$ belonging to the input domain [34]. Even though this strategy should allow to obtain a copy of the oracle, it makes the attack untractable. Thereby they introduced a heuristic to limit the number of queries to the target model in the first step of the two-step strategy, a data generation technique called *Jacobian-based Dataset Augmentation*. The objective is to collect a dataset that allows to approximate the oracle's decision boundaries with a few queries. The intuition is

that it's necessary to look at the directions in which the model's output is varying, around the initial training points [34]. The surrogate architecture F will require more predictions in these directions to approximate better the oracle O. These directions are established using the Jacobian matrix J_F of the surrogate model F, that's the matrix of the first partial derivative of the loss function of the DNN F. The matrix is evaluated at several points $\vec{x}$. In particular, the sign of the J_F dimension that corresponds to the label $\tilde{O}(\vec{x})$ is evaluated and then $\vec{x}$ is perturbed according to the sign [34].
The strategy proposed is articulated as follows. Firstly, the attacker has to collect a very small set S_0 of instances of the input domain, not necessary from the same distribution from which the target model was trained [34]. Also, she has to choose an appropriate surrogate model, as discusses in section 3.3.1. Then, the attacker proceeds to the first step of the two-step attack strategy, that's the substitute training, describe by the Algorithm 2. λ is the step size that specifies the amount of perturbation, while max_ρ is the maximum number of substitute training epochs decided at priori. Firstly, each instance in $\vec{x}$ in the surrogate training set is labeled by O. Then F is trained on D and finally the initial substitute training set S_ρ is augmented, producing the next substitute training set $S_{\rho+1}$. In order to improve the quality of the surrogate model, they introduced also an alternation period τ after which the step size is multiplied by -1, so the step size λ becomes:

$$\lambda_\rho = \lambda \cdot (-1)^{\left\lfloor \frac{\rho}{\tau} \right\rfloor}$$

Moreover they decided to apply reservoir sampling to reduce the number of queries: after the training epoch σ, only k instances are selected from the set S_ρ to augment the surrogate training set. Then, the total number of queries submitted to the target is $n \cdot 2^\sigma + k \cdot (\rho - \sigma)$, where n is the number of samples in S_0 [34].
After the max_ρ training epochs, F is used to generate adversarial examples using FGSM. Then the adversarial examples are submitted to the target model to test if they transfer.
They tested their attack strategy against a classifier of Amazon Machine Learning[7] and Google's Cloud prediction API[8] trained on the 50000 instances of the MNIST training set (a dataset of handwritten digit that we also use in the experimental evaluation, Chapter 5). They started the procedure with $n = 100$ and obtained a misclassification rate of 96% and 91% with 6 training epochs, using only 2000 queries for collecting predictions for the training set of the surrogate model. They produced adversarial examples by perturbing the 10000 instances of the test set MNIST using FGSM with $\epsilon = 3$, then the number of queries spent for the second step is not included in the attacker's budget B.

Criticalities

Although the two-step attack strategy presented, among with the optimization of the training phase, shows great results, the approach presents some criticalities:

- The particular implementation of the strategy requires always to establish a priori the number of training epochs max_ρ. Even though the attacker can use the Jacobian-based Dataset Augmentation to reduce the number of

[7]https://aws.amazon.com/machine-learning
[8]https://cloud.google.com/prediction/

queries, she has to fix the value of the parameter before the attack. Papernot et. al. [34] don't show a heuristic for establishing the optimal value of the parameter $\max_\rho$, i.e., when to switch from step 1 to step 2 of the strategy. It was only noted that an high number of training epochs doesn't necessarily improve the transferability of the advexps crafted.

- They don't consider the number of queries spent to submit adversarial examples as an part of the budget B spent.

The first problem of their proposal is also a criticality of the traditional two-step attack strategy. In the next chapter, the drawbacks of the traditional two-step attack strategy are treated in detail and a strategy to dynamically adapt the number of queries of the first phase is presented.

Chapter 4

Adaptive black-box evasion attack

Black-box evasion attacks against machine learning models are, in practice, difficult, because of the limited knowledge about the target model and the restricted query access. Transferability allows to overcome the lack of information, but not the limited number of queries.
In the previous chapter, we presented the traditional two-step attack strategy and a particular implementation by Papernot et. al. [34]. In this chapter, we deepen some criticalities of the traditional approach when deployed in the setting with a limited number of queries available. Indeed, the attacker has to choose carefully whether to exploit a query either to gather information about the target classifier in order to improve the quality of the surrogate model or to attempt an evasion attack. The tension between these two actions, due to the limited budget, induces a trade-off in the usage of the queries. For this reason, we proposed a new transfer-based two-step attack strategy for performing evasion attacks in the black-box setting. The strategy presented is adaptive, in the sense that it alternates the two possible actions depending on the actual knowledge of the attacker.
The chapter is structured as follows. In section 4.1, we describe some critical aspect of the traditional two-step attack strategy, in particular the inherent tension between the two steps. Then, in section 4.2 we focus on AMEBA, the first adaptive approach to the black-box generation of evasion attacks against ML models.

4.1 The Inherent Tension in the Two-steps Evasion Attack

In the black-box setting with limited query budget, the attacker is supposed to have a maximum number B of queries to the target model. In this setting, the traditional two-step attack strategy should be deployed carefully.
We can observe that there is a *tension* between the two steps of the attack strategy. On the one hand, the attacker needs to query the target model multiple times in order to disclose its behavior and train a faithful surrogate model. On the other hand, the attacker wants to query the target model with as many evasion attacks as possible to maximize the number of misclassifications and transferred adversarial examples. The traditional two-step attack strategy handles this tension using two assumptions:

- The number of queries for the surrogate training step and the evasion step are decided before the execution of the attack.

- The two steps are independent and separated.

However, these two assumptions limit the actions of the attacker and they don't allow to exploit better the limited budget. Indeed the attacker may want to automatically adapt the use of queries using the information collected during the execution of the attack. For example, she may want to switch from one step to the other and vice versa. For this reason the strategy is sub-optimal. The optimal transfer-based two-step strategy in the limited query setting is far from straightforward. Three research questions emerge:

- When shall the attack strategy switch from step 1 to step 2?
- Why should the attack strategy switch from step 1 to step 2 without the chance of switching back and forth a certain number of times?
- Why should the attacker follow a fixed two-step strategy and not resort to a more sophisticated approach which dynamically learns how to behave?

In this work, we propose to move away from the two-step attack strategy of previous work and we present a new *adaptive* attack strategy, which dynamically learns whether queries to the target model should be leveraged for surrogate model training (step 1) or for evasion attack crafting (step 2). It makes black-box evasion attacks more effective and practical by automatically dealing with the delicate tension discussed above.

4.2 Adaptive Attack Strategy

In this section, AMEBA is described. In particular, we discuss the threat model considered, we operate the reduction from the two-step evasion attack problem to the MAB problem and we formulate the adaptive approach by exploiting the Thompson sampling algorithm, used to solve the MAB problem. A detailed description of the AMEBA algorithm is provided among with the discussion of how to treat certain implementation details.

4.2.1 Threat Model

Attack Setting

We consider an attacker whose goal is to craft successful evasion attacks against a target model h. The attacker has only black-box access to it and she can perform a limited number of queries asking for class predictions of chosen instances. The attacker has a surrogate model $\hat{h}$ that has to be trained, but it can be used to craft evasion attacks. The objective of the attacker is to maximize the number of successful evasion attacks against the target h given the limited number of queries available.
We suppose that the attacker has access to three datasets and she can exploit two possible actions. In Figure 4.1 we sum up these assumptions. The attacker has access to the following datasets:

- $\mathcal{D}_{trn}$: a set of instances $\{(\vec{x}_i, h(\vec{x}_i))\}$ labeled with the class predictions of the target h, used for surrogate model training. For example, $\mathcal{D}_{trn}$ might be a collection of known spam and ham messages available to the attacker.

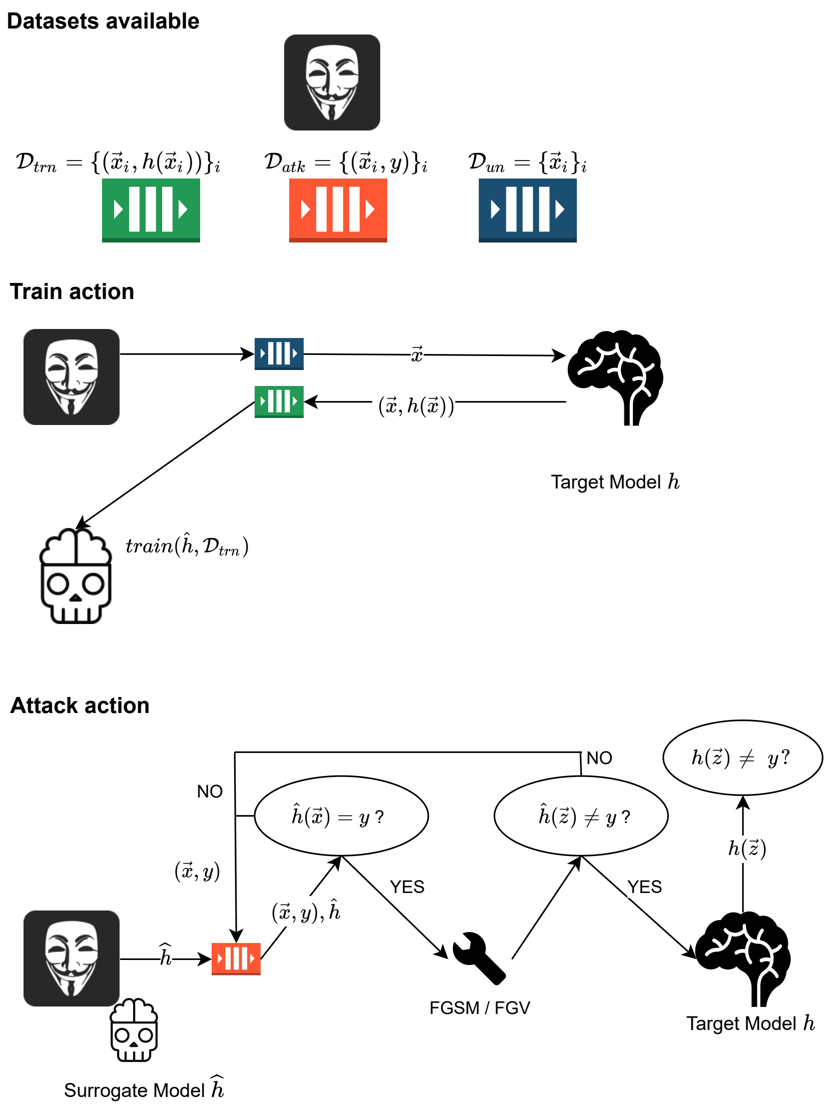


Figure 4.1: Threat model description

- $\mathcal{D}_{atk}$: a set of instances $\{(\vec{x}_i, f(\vec{x}_i))\}$ labeled with their true labels, used for evasion attack crafting. For example, $\mathcal{D}_{atk}$ might be a set of spam messages that the attacker wants to evade a spam filter.

- $\mathcal{D}_{un}$: a set of unlabeled instances $\{\vec{x}_i\}$, used to collect additional class predictions from the target h. For example, $\mathcal{D}_{un}$ might include messages that the attacker has written himself, whose class predictions would be unknown.

Initially are not made any assumption on the instances in these three datasets: they can be equal, overlapping or disjoint. A convenient representation of these threee datasets is the queue model, i.e., they have standard push and pop operations.

Two Steps Attack Strategy

In order to perform the evasion attack, the attacker has initally to train the surrogate $\hat{h}$ using $\mathcal{D}_{trn}$, then she has to exploit the budget available to both augment $\mathcal{D}_{trn}$ using the predictions of the target of unlabeled instances and to attempt evasion attacks against the target model h using the transferability property of the crafted attacks against $\hat{h}$. More specifically, the attacker operates by choosing between the two possible actions:

1. *Train*: the attacker pops an instance $\vec{x} \in \mathcal{D}_{un}$, queries h to learn the prediction $y = h(\vec{x})$ and extends $\mathcal{D}_{trn}$ with $(\vec{x}, y)$. The attacker then updates the surrogate $\hat{h}$ by retraining it over the extended $\mathcal{D}_{trn}$.

2. *Attack*: the attacker pops an instance $(\vec{x}, y) \in \mathcal{D}_{atk}$. If $\hat{h}(\vec{x}) = y$, the attacker uses $\vec{x}$ to craft an evasion attack $\vec{z}$ against $\hat{h}$ by using an appropriate attack strategy, e.g., FGV. If a successful evasion attack is found, i.e., if $\hat{h}(\vec{z}) \neq y$, the attacker submits $\vec{z}$ to the target model h and verifies whether $h(\vec{z}) \neq y$.

The attacker stops when the maximum number of queries B to the target model h has been performed. Note that each action requires only one query to be performed. However, the *Attack* action might fail without querying h in two cases, both due to the definition of evasion attack stated in section 2.3.3:

- $\vec{x}$ is misclassified by $\hat{h}$. Indeed, the definition of evasion attack assumes that the base instance $\vec{x}$ is correctly classified by the surrogate model $\hat{h}$.

- It is impossible to turn $\vec{x}$ into a successful evasion attack against $\hat{h}$. The perturbation constraint $||\vec{z} - \vec{x}||_p \leq \epsilon$ of the evasion attack, where p indicates the norm type, limit the extent at which the instance $\vec{x}$ can be perturbed.

In order to handle these two cases, two different choices are possible. The instance $\vec{x}$ might be discarded, losing an effective potential adversarial example and consuming instances without giving another chance. However, the continuous training process of the surrogate $\hat{h}$ could both improve the accuracy of the target model and enable new attacks. Indeed the decision boundaries induced by the surrogate $\hat{h}$ on the feature space change as new instances are added to the training set, making possible to obtain a valid adversarial example from an instance that was previously unusable. Then, in both cases of failure, it's assumed that $(\vec{x}, y)$ is only temporarily discarded and pushed back into $\mathcal{D}_{atk}$ for later use.
The attacker may try to interleave dynamically the two actions *Attack* and *Train* in order to minimize the number of queries used for augmenting $\mathcal{D}_{trn}$ and to maximize the number of generated evasion attacks against the target h. The outcomes of the two actions are inevitably linked: if the use of the *Train* action allows to improve the quality of surrogate model in terms of similarity to the target model, then also the transferability of the evasion attacks and the number of successful evasion attacks will improve.
Then a trade-off emerges between spending queries for one action or for the other one, since the *Train* action has to be exploited to obtain a high quality surrogate model and maximize the transferability, but it should not be performed too many times. Indeed an abuse of the *Train* action is not justified by the empirical evidence, since there exists a limit number of instances in $\mathcal{D}_{trn}$ over that the alignment between surrogate and target model doesn't improve, as state by Papernot et. al. [34].

4.2.2 Reduction to MAB

The Reduction

The best attack strategy could be obtained by reducing the problem to the MAB problem. The two available actions are $\mathcal{A} = \{\mathit{Train}, \mathit{Attack}\}$ and, supposing that the attacker executes one query for each action, the number of rounds T of the MAB problem coincides with the attacker's budget B. Since the model adopted is the Bernoulli MAB problem, the two actions are supposed to provide a binary reward, i.e. success or failure. The definition of the reward of the two actions is the key aspect in order the make the reduction coherent with the attacker's goal, so that the maximization of cumulative reward of MAB matches the maximization of the number of successful attacks.
The *Train* action should give the success reward when the enrichment of the surrogate training set $\mathcal{D}_{trn}$ has effectively improved the quality of the surrogate model, so that the similarity between h and $\hat{h}$. However, the similarity between the two models cannot be defined on the basis of the single query performed during the action, for example using the matching of the predictions of the surrogate and target model on the instance, because of the selection bias introduced by using the single instance for the assessment. In the ideal case, the best choice would be to measure the similarity using the accuracy of $\hat{h}$ on another dataset labelled by h, but in this setting it's infeasible because of the limited budget B. Since $\mathcal{D}_{trn}$ is built of instances labelled by h, then the 10-fold cross validation accuracy score of $\hat{h}$ on $\mathcal{D}_{trn}$ could be used as a proxy of the similarity between the two models. Indeed it gives an overall summary of the test error of $\hat{h}$ on a set of instances labelled by h. Given that, the success of the *Train* actions is defined as an improvement of the similarity between surrogate and target model computed on $\mathcal{D}_{trn}$ before and after the inclusion of the new labelled instance in $\mathcal{D}_{trn}$.
On the other hand, the definition of the notion of success for the *Attack* action is straightforward, since it should be considered successful only when the crafted adversarial examples on $\hat{h}$ transfers to h. Then, more formally, the *Attack* action is successful when the crafted evasion attack $\vec{z}$ against h from the instance $\vec{x}$ with label y is so that $\hat{h}(\vec{z}) \neq y$.

Details about the Actions

In practice, the attacker may face extreme situations. The *Train* action always lead to perform one query to the target model, since there aren't constraints about which instance should be submitted. However, the *Attack* action may not perform any query to h when the evasion attack against the surrogate $\hat{h}$ fails.
Two subtle problems arise from this consideration. Firstly, it's not correct to consider the failure in the adversarial example crafting as a failure, since the semantics of the success and failure of the *Attack* action is strictly related to the transferability of the crafted evasion attack. Secondly, the potential disconnection between the total number of times in which the actions are performed and the number of submitted queries make the reduction wrong, since the MAB problem supposes that at each turn an action is performed and, in the reduction proposed, a turn is carried out only when the budget B is decreased, i.e., a query is submitted. To close this gap, it's assumed that the failure does not discourage the attacker, who just moves to the next instance of $\mathcal{D}_{atk}$ until a successful evasion attack against $\hat{h}$ is found. The side-effect of this choice consists in giving some instances a second

chance in a future round. This is reasonable, since the *Train* action is supposed to increase the similarity between the surrogate $\hat{h}$ and the target h, enabling new attacks on instances in $\mathcal{D}_{atk}$ previously misclassified or not suitable as starting point for crafting an adversarial example. If the attacker cannot craft any successful evasion attack against $\hat{h}$ using the instances in $\mathcal{D}_{atk}$, we assume that the *Train* action is taken instead. This is the only option available to the attacker given the current surrogate model and at the same time this choice ensures the invariant that each action consumes exactly one query to the target model h.

The Dependence between the Reward Distributions

MAB problems assumes that the reward distributions D_a of the actions are permanent and independent. However, since the surrogate model $\hat{h}$ is updated after every *Train* action and its quality affects the probability of success of both actions, the two Bernoulli distributions of the rewards of the two actions cannot be supposed independent. Yet, the experimental evaluation (Section 5) shows the effectiveness of the Thompson sampling algorithm, which is due to the fact that the reward distributions do not negatively interfere, but rather boost one another as discussed.

The Justification

We have chosen to reduce the problem to the Bernoulli MAB since the trade-off between the two different ways to spend the queries is similar to the exploration-exploitation trade-off in the Reinforcement Learning problems (Section 2.4.1). On the one hand, the attacker has to explore the outcome of the two possible actions to investigate which is the most profitable. On the other hand, the attacker should exploit the most convenient action using the knowledge gathered during the attack in order to reach her goal. Moreover, the reduction to the Bernoulli MAB problem allows to exploit one of its solving algorithm to automate the expected behaviour of the attacker in the different scenarios she may face during the attack.
Secondly, we propose the reward scheme stated above because it's effective. Indeed, when the *Attack* success rate is low, it's better to exploit the *Train* action in order to improve the quality of the h model in terms of similarity to the $\hat{h}$. Iteration after iteration, the success rate of the *Train* action decreases, since the similarity between the two models reaches a plateau. Then the *Attack* action becomes the most valuable choice.

4.2.3 AMEBA

Once the reduction from the two-step attack strategy problem and the MAB is established, the Thompson sampling algorithm is one of the suitable choices to provide a solution to the MAB problem and to implement the adaptive attack strategy. The details of the resulting attack strategy, called AMEBA (Adversarial Multi ArmEd BAndit), are formalized in Algorithm 3. Although the pseudocode relies on FGV for evasion attack crafting, any other algorithm for the same task could be used.
The algorithm starts by initializing the pseudo-counts of successes and failures of the two actions (lines 1–2). Then, at each of the T rounds, so at each of the B queries allowed, it selects the action with the highest estimate of success. In

lines 8–25, the *Attack* action is implemented as discussed in the previous Section 4.2.2. First, the attacker iterates through $\mathcal{D}_{atk}$ in search of an instance $(\vec{x}, y)$ for which it is possible to craft a successful evasion attack against the surrogate model $\hat{h}$. If such an attack $\vec{z}$ is found (line 21), it is submitted to the target model gaining a reward $r_t = 1(h(\vec{z}) \neq y)$, where $1(p)$ equals 1 if the predicate p is true and 0 otherwise. In other words, $r_t = 1$ if h misclassifies the perturbed instance $\vec{z}$ and $r_t = 0$ otherwise.
If no evading instance is found, or if the estimated probability of attack success $\hat{\theta}_{Attack}$ is not greater than train success $\hat{\theta}_{Train}$, then the *Train* action is performed (lines 26–35). In this case, a new instance $\vec{x}$ retrieved from $\mathcal{D}_{un}$ is submitted to the target model h to get the corresponding prediction and then used to enrich the training set $\mathcal{D}_{trn}$. A new surrogate model is finally trained and used in the subsequent iterations. We set the reward to 1 if we observe an increase of the cross-validation score of the surrogate, to 0 otherwise.

Algorithm 3 The AMEBA attack strategy

```
1: for $a$ in $\{Train, Attack\}$ do
2:     $(S_a, F_a) \leftarrow (1, 1)$                              ▷ Initialization
3: end for
4: for $t = 1, ..., T$ do
5:     Sample $\hat{\theta}_{Train} \sim Beta(S_{Train}, F_{Train})$
6:     Sample $\hat{\theta}_{Attack} \sim Beta(S_{Attack}, F_{Attack})$
7:     $evading \leftarrow \perp$                              ▷ Evasion attack yet not found
8:     if $\hat{\theta}_{Attack} > \hat{\theta}_{Train}$ then                              ▷ *Attack* action
9:         $available \leftarrow |\mathcal{D}_{atk}|$
10:        while $evading = \perp \wedge available > 0$ do
11:            $available \leftarrow available - 1$
12:            $(\vec{x}, y) \leftarrow \textsc{Pop}(\mathcal{D}_{atk})$
13:            $\vec{z} \leftarrow \text{FGV}((\vec{x}, y), \hat{h})$                              ▷ Craft evasion attack
14:            if $\hat{h}(\vec{x}) = y \wedge \hat{h}(\vec{z}) \neq y$ then                              ▷ Confirm evasion
15:                $evading \leftarrow (\vec{z}, y)$
16:            else
17:                $\mathcal{D}_{atk} \leftarrow \textsc{Push}(\mathcal{D}_{atk}, (\vec{x}, y))$
18:            end if
19:        end while
20:    end if
21:    if $evading \neq \perp$ then                              ▷ Found evasion attack on $\hat{h}$
22:        $a_t \leftarrow Attack$
23:        $(\vec{z}, y) \leftarrow evading$
24:        $r_t \leftarrow 1(h(\vec{z}) \neq y)$                              ▷ Check transferability on $h$
25:        $(S_{a_t}, F_{a_t}) \leftarrow (S_{a_t} + r_t, F_{a_t} + (1 - r_t))$
26:    else                              ▷ *Train* action
27:        $a_t \leftarrow Train$
28:        $old\_cv\_score \leftarrow \textsc{CrossValScore}(\hat{h}, \mathcal{D}_{trn})$
29:        $\vec{x} \leftarrow \textsc{Pop}(\mathcal{D}_{un})$
30:        $y \leftarrow h(\vec{x})$
31:        $\mathcal{D}_{trn} \leftarrow \textsc{Push}(\mathcal{D}_{trn}, (\vec{x}, y))$
32:        $\hat{h} \leftarrow \textsc{Train}(\mathcal{D}_{trn})$
33:        $r_t \leftarrow 1(\textsc{CrossValScore}(\hat{h}, \mathcal{D}_{trn}) > old\_cv\_score)$
34:        $(S_{a_t}, F_{a_t}) \leftarrow (S_{a_t} + r_t, F_{a_t} + (1 - r_t))$
35:    end if
36: end for
```

Chapter 5

Experimental evaluation

In this chapter we describe the setup and the results of the experimental evaluation of AMEBA. We simulated different attack scenarios in order to show that AMEBA is appropriate for pratical usage.
In section 5.1 the experimental setup is presented, i.e. the datasets, the surrogate and target models and the perturbations used to craft adversarial examples. Next, in section 5.2 the adopted methodology is explained, in particular how the sets needed to the attack are built and how AMEBA is compared against the traditional two-step attack strategy, our baseline. Section 5.3 presents the results of our experimental evaluatuation, while in section 5.4 we present the time performance of AMEBA, in order to undestrand if the complete evasion attack can be carried out up to query budget exhaustion in a reasonable time. Finally in section 5.5 are discussed the key points that allow AMEBA to be effective.

5.1 Experimental Setup

In this section we describe the two experimental settings adopted to experimentally evaluate AMEBA. The two settings are mainly differentiated using the type of classification task associated to the datasets used. In each setting, a range of classifiers are used as surrogate and target models, in order to reproduce different threats and application settings and show that AMEBA is effective in general.

5.1.1 Datasets

The experimental evaluation of AMEBA is performed on four public datasets. The statistics of the datasets are summarized in Table 5.1, where we report the number of instances, the number of features, the number of classes and the class distributions, positive vs negative instances in the case of datasets with two label. In particular, we use:

- Spambase[1], a dataset of spam and no spam emails gathered by the Hewlett-Packard Labs. This dataset is associated with a binary classification task, the classification of spam emails. Legitimate emails are labeled as 0, while spam emails are labeled as 1. Each email is described by 57 features: the first 48 are the percentage of appearance of particular words, 6 features represent the percentage of appearance of particular characters, feature 55 represents

[1]https://archive.ics.uci.edu/ml/datasets/Spambase

Table 5.1: Statistics of the datasets used for binary classification tasks

	Spambase	**Wine**	**CodRNA**	**MNIST**
n. of instances	4601	6495	488565	60000
n. of features	54	12	8	784
n. of classes	2	2	2	10
class distrib.	$39 \div 61$	$25 \div 75$	$67 \div 33$	10% for each class

the average length of uninterrupted sequences of capital letters, feature 56 represents the length of longest uninterrupted sequence of capital letters and finally feature 57 represents total number of capital letters.

- Wine Quality[2], dataset related to red and white variants of a Portuguese wine. This dataset is associated to a binary classification task that consists in discriminating the red (1) and white (0) wine colors. Each wine is described by 11 physicochemical variables.
- CodRNA[3], dataset that contains sequences of RNA described by 8 features. The classification task associated with this dataset is binary since the sequences are labeled as negative (0) or positive (1).
- MNIST[4], dataset of handwritten digits in 28x28 pixels grey scale images. The classification task associated with this dataset is multiclass, since the class labels are the 10 digits.

5.1.2 Setup for Binary Classification Tasks

Classifiers used

For the datasets associated with a binary classification task, a Linear SVM model is used as surrogate model to craft black-box evasion attacks against three different target models available in the state-of-the-art scikit-learn library [35]:

1. a decision tree ensemble model learned using the Random Forest algorithm, a non-differentiable model;
2. a decision tree ensemble model learned using the Adaboost algorithm, a non-differentiable model;
3. a logistic regression classifier, a differentiable model that presents some similarities with Linear SVM (see Section 2.2.4).

Training

All the models are trained after normalizing features in the interval $[0, 1]$. The use of feature normalization is a standard technique used in many other works in literature and it prevents the negative effect that different measurement scales of the features have on non-robust classifiers like Linear SVM.

[2]https://archive.ics.uci.edu/ml/datasets/wine quality
[3]https://www.openml.org/d/351
[4]http://yann.lecun.com/exdb/mnist/

The training procedure is preceded by the hyperparameter tuning of the models. The hyperparameter tuning process consists in choosing the best set of hyperparameter values for each learning algorithm. In this work it's conducted by using the grid search strategy with the 10-fold cross validation accuracy on the training set as performance measure. In particular, the training set is firstly partitioned into 10 different subsets. Then the cross validation accuracy is computed as the average of the 10 accuracies scores retrieved by training the classifier on all the partitions except one, that's used as validation set. The procedure is repeated for every combination of hyperparameters value to detect the combination that allows the classifier to provide the best accuracy.
The hyperparameters tested are:

- Random Forest: number of trees ($\{64, 128, 256, 512, 1024, 2048, 3072\}$) and node splitting criterion (gini impurity and max entropy);
- AdaBoost: number of trees ($\{64, 128, 256, 512, 1024\}$), node splitting criterion (gini impurity and entropy) and maximum number of leaves ($\{8, 16, 32, 64\}$);
- logistic regression: regularization factor C (from 10^{-4} to 10^{4}).

Justification of Linear SVM as Surrogate Model

The choice of using Linear SVM as surrogate model is mainly motivated by previous work, which showed that such type of model allows to generate effective black-box attacks with strong transferability [17]. Also our experiments confirm that Linear SVM is a quite effective surrogate model. Moreover, it's very simple to craft evasion attacks against a Linear SVM using algorithms like the FGV variant discusses in section 2.3.4.
The regularization factor of the SVM is $C = 0.50$, resulting in an highly regularized model, choice motivated by the observation that highly regularized models typically provide better transferability [17]. Other surrogate models are not considered since Demontis et. al. [17] shown that an highly regularized Linear SVM allows to obtain transferabilities against a wide range of models that are on average almost comparable to the ones obtained by using other surrogate models.

Evasion Attack Crafting Algorithm

Adversarial examples are crafted using the algorithm 4 called $\text{FGV}_{\text{minpert}}$, that exploit the FGV attack. The strategy consists in finding the minimum perturbation required for crafting a successful evasion attack against the targeted model (the surrogate model) among a grid of increasing perturbation values, i.e., using line search. For simplicity, the objective of the attack isn't to evade a specific class, i.e. the attack considered is the *untargeted*.
The algorithm iterates on the sequence of admitted perturbations (line 1). The perturbation ϵ' ranges from a user defined perturbation $\epsilon_0 \in \mathbb{R}$ to another user defined maximum perturbation $\epsilon \in \mathbb{R}$, with step size $s \in \mathbb{R}$. An evasion attack $\vec{z}$ is crafted against the target model (the surrogate model $\hat{h}$) and the instance $(\vec{x}, y)$ using perturbation ϵ' (line 2). If $\vec{z}$ evades $\hat{h}$, then the crafting algorithm can be stopped, since the adversarial example requiring the minimum perturbation is found (line 4). In the other case, the algorithm continues to examine other possible perturbations.

Algorithm 4 FGV$_{\text{minpert}}$

```
1: for $\epsilon' = \epsilon_0, \epsilon_0 + s, \epsilon_0 + 2s, \ldots, \epsilon$ do
2:     $\vec{z} \leftarrow \text{FGV}((\vec{x}, y), \hat{h}, \epsilon')$          ▷ Craft the evasion attack
3:     if $\hat{h}(\vec{z}) \neq y$ then
4:         break                          ▷ Exit if a valid advexp has been found
5:     end if
6: end for
7: return $\vec{z}$
```

Attack Scenarios

In order to consider a wide range of attack scenarios, we decide to examine the cases in which the attacker can perform at most $T = 1000$ and $T = 2000$ queries to the target model. Testing two attacker's budgets allows to show that the results obtained are not biased by the chosen number of queries.
Moreover, to further differentiate the attack scenarios, the attacker uses different maximum perturbation magnitudes that depend on the specific dataset from which are taken instances to generate adversarial examples. In particular, $\epsilon = 0.10,\ 0.15$ is used with Spambase and CodRNA and $\epsilon = 0.20,\ 0.25$ for Wine, while s is set as 0.01 for all the datasets.

5.1.3 Setup for the multiclass classification tasks

Since deep learning models are increasingly used for perceptual tasks like image classification and are also primary target in the adversarial machine learning literature [6], we also carry out additional experiments with the MNIST dataset. The results obtained allow to show that our proposal generalizes also to deep learning.

Classifiers used

We train four deep learning models from the literature as targets. The models are taken from [44] and its shared code.[5] Specifically, we train the following targets:

- a standard Convolutional Neural Network (CNN), which is a simple model for image classification;
- the Model A and Model C networks, which are more sophisticated models exhibiting near-perfect accuracy on MNIST;
- a variant of the Model A network where we remove the drop-out layers. Since drop-out layers provide robustness against noise, we might expect this network to be more vulnerable to evasion attacks than Model A.

The Caffe[6] variant of a traditional LeNet [30] network is used as surrogate in all cases. This model has roughly the same complexity of the target CNN in terms of number of parameters to train, while being significantly smaller than the other target models. In Table 5.2 are reported the architectures of all the networks.

[5]https://github.com/suyeecav/Hybrid-Attack

[6]https://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/01-learning-lenet.ipynb

CNN	**Model A**	**Model C**	**LeNet**
Conv(32, 3, 3) + Relu	Conv(64, 5, 5) + Relu	Conv(128, 3, 3) + Relu	Conv(20, 5, 5) + Relu
Conv(64, 3, 3) + Relu	Conv(64, 5, 5) + Relu	Conv(64, 3, 3) + Relu	MaxPool(2,2)
MaxPool(2,2)	Dropout(0.25)	Dropout(0.25)	Conv(50, 5, 5) + Relu
Dropout(0.25)	FC(128) + Relu	FC(128) + Relu	MaxPool(2,2)
FC(128) + Relu	Dropout(0.5)	Dropout(0.5)	FC(500) + Relu
Dropout(0.5)	FC(10) + Softmax	FC(10) + Softmax	FC(10) + Softmax
FC(10) + Softmax			

Table 5.2: Neural network architectures used in this work.

Training

The target models are trained for a maximum number of 200 epochs with Adam stochastic optimization [29], a learning rate of 10^{-3} and batch size 128. The effective number of epochs is selected by using early stopping: training process is stopped after 50 epochs in which the error on the validation set of 10000 randomly sampled instances has not improved. All target models achieve at least 99% accuracy on a randomly sampled test set of 10000 instances. The surrogate model is trained for 15 epochs with Adam optimizer, a learning rate of 10^{-3} and batch size 32. In order to prevent overfitting, we train both the target models and the surrogate using data augmentation, with a random rotation of ± 20 degrees of the digit at most, a random right and left shift of 0.2 of the total width of the image at most and a random zoom in the range 0.8-1.2 (the value 1 leaves the instance unmodified). The choice of using data augmentation also for the surrogate training set is reasonable, since the attacker is allowed to use any method to expand the training set of the surrogate model.

Evasion Attack Crafting Algorithm and Attack Scenarios

In this scenario, the standard FGV is used, as described in 2.3.4, with a fixed perturbations $\varepsilon = 2.0$ and $\varepsilon = 3.0$. This choice allows to obtain reasonable transferability values and to create images which are still recognizable by humans, as the examples of perturbed images in Figure 5.1 show. Moreover, the similarity between the surrogate and the target model is computed using 5-fold cross validation instead of the 10-fold cross validation of the previous setting. Finally, we assume the attacker can perform $T = 3000$ queries to the target model. We assign a larger budget to the attacker in this experiment, because neural network models are more complex and require more data to be trained.

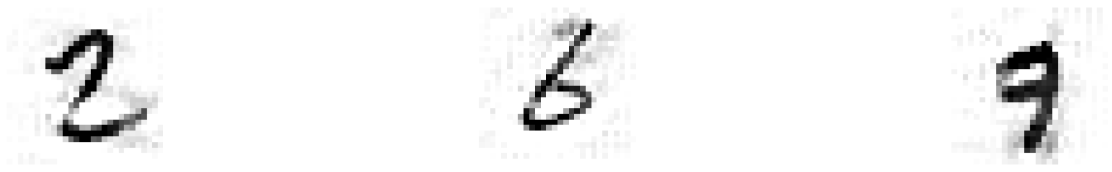


Figure 5.1: Examples of perturbed images for = 3.0

5.2 Methodology

The methodology adopted in the experiments is set up as follows.

	Hyperparameters		
Model	**Spambase**	**Wine**	**CodRNA**
Random Forest	n_estimators = 128, criterion = "entropy"	n_estimators = 256, criterion = "gini"	n_estimators = 3072, criterion = "entropy"
AdaBoost	n_estimators = 64, criterion = "gini", max_leaf_nodes =32	n_estimators = 512, criterion = "gini", max_leaf_nodes = 8	n_estimators = 1024, criterion = "gini", max_leaf_nodes = 8
Logistic Regression	C=100	C=100	C=1000

Table 5.3: Hyperparameter configuration of the target models for the binary classification tasks

	Spambase	**Wine**	**CodRNA**
Random Forest	0.96	0.99	0.97
AdaBoost	0.97	0.99	0.97
Logistic Regression	0.93	0.99	0.95

Table 5.4: Accuracy scores of the target models for the binary classification tasks

Datasets splitting

Each dataset $\mathcal{D}$ is partitioned into two sets $\mathcal{D}_{tgt}$ and $\mathcal{D}_{sur}$ using stratified random sampling, i.e. they are randomly partitioned maintaining the original class distribution. The $\mathcal{D}_{tgt}$ set is used to train the target models. Due to the small number of instances of Spambase and Wine, only 1600 instances of Spambase and 3000 instances of Wine are reserved for it, while 100000 instances of the CodRNA dataset and and 40000 instances of MNIST are reserved.

Actual classifiers used

The best value for the hyperparameters of the target models for the binary classification tasks are reported in Table 5.3. The regularization factor C of the Logistic Regression classifier is 100 or 1000 across the different training sets, while the number of estimators of the ensemble models results to be greater with the CodRNA dataset than with the Spambase dataset.
The amount of instances used in the training sets for the target models and the hyperparameter values chosen allow to achieve high values of accuracy with all the target models, as reported in Table 5.4. In particular, the 10-fold cross validation accuracy values of the target models involved in the various datasets are shown. All the models reaches at least 93% accuracy on Spambase, Wine and CodRNA, then the choice of using small $\mathcal{D}_{tgt}$ from Spambase and Wine seems reasonable. The classification of spam and bulk emails seems the most difficult task, since the target models reach the lowest accuracy, while all the three models perform very well on Wine, with an accuracy of 99%. About the target CNNs used for the multiclass classification task, the hyperparameters like batch size and learning rate where decided in advance. All the target CNNs reaches 99% of accuracy on the sampled MNIST test set.

Sets available to the attacker

The $\mathcal{D}_{sur}$ set, instead, is used to train the surrogate model and craft evasion attacks. In particular, it's partitioned into the sets $\mathcal{D}_{trn}$, $\mathcal{D}_{atk}$ and $\mathcal{D}_{un}$ again using stratified random sampling. More precisely, in the experiments with budget $T = 1000$, these sets are constructed as follows:

- $\mathcal{D}_{trn}$ includes 100 instances used to train the initial surrogate model. The labels are provided by the target model, then 100 queries are spent. The initial training set is necessary in order to train the surrogate model in the first stages of the AMEBA attack strategy. The choice of size 100 is reasonable and suitable for obtaining a minimally meaningful surrogate model, as the experimental results will demonstrate.

- $\mathcal{D}_{atk}$ is made up of 900 instances correctly classified by the target models, which are used to craft evasion attacks. Even if the correct classification of the instances used to generate the attacks is not a prerequirement, an instance classified uncorrectly represents a trivial adversarial example, so the instances manipulated are considered to be correctly classified by the target as in previous works. The size of $\mathcal{D}_{atk}$ is motivated by the fact that the maximum number of evasion attacks that the attacker can perform against the target model is limited by the budget, 1000, and by the number of queries used to label the initial $\mathcal{D}_{trn}$, 100.

- $\mathcal{D}_{un}$ includes the remaining (unlabeled) instances of $\mathcal{D}_{sur}$, which are used to augment $\mathcal{D}_{trn}$ with the class predictions from the target model and improve the quality of the surrogate. Specifically, $\mathcal{D}_{un} = \{\vec{x} \mid \exists y : (\vec{x}, y) \in \mathcal{D}_{sur} \setminus (\mathcal{D}_{trn} \cup \mathcal{D}_{atk})\}$. The usage of unlabelled instances is coherent with respect to a real scenario, in which labelling instances independently could be expensive and requires using a service like Automated Data Labeling by Amazon[7].

For budget $T = 2000$ and $T = 3000$, the size of $\mathcal{D}_{atk}$ is 1900 and 2900 respectively.

Comparison against the baseline strategy

AMEBA is compared against a traditional two-step attack strategy, where the separation between the training phase (step 1) and the attack phase (step 2) is clear. Moreover, prior work assumed the usage of a reasonably accurate surrogate model in phase 2, learned using the instances gathered in phase 1 through a fixed number of training rounds [34]. However it doesn't investigate how to fix the number of training rounds. Yet this is a delicate point, given that the number of available queries to the target model is limited.
The comparison is then realized by implementing the two steps attack strategy using multiple baselines, in order to cover multiple possible choices for the number of queries spent for surrogate model training. In particular, for each $i \in \{0, 50, 100, \ldots, 700\}$ with $T = 1000$, $i \in \{0, 100, 200, \ldots, 1700\}$ with $T = 2000$ and $i \in \{0, 200, 400, \ldots, 2600\}$ with $T = 3000$, i instances are collected from $\mathcal{D}_{atk}$ using stratified random sampling and are added to $\mathcal{D}_{trn}$, generating datasets $\mathcal{D}^{i}_{trn}$ (of size $i + 100$). The remaining instances in $\mathcal{D}_{atk}$ are used for evasion attack crafting. Less than 200 instances are never used as $\mathcal{D}_{atk}$, otherwise the available attack instances would be so few to significantly lower the attack opportunities. In the worst comparison setting, a powerful clairvoyant attacker would choose in advance the best training size for the surrogate model, and therefore perform as the best of the considered baselines. Then the objective is to show that AMEBA

[7]https://docs.aws.amazon.com/sagemaker/latest/dg/sms-automated-labeling.html

approximates the best baseline or even improves over it without having to detect at priori which is the best splitting.

Performance measures

The performance of the attack strategies is evaluated in terms of the following two measures:

1. the absolute number of successful evasion attacks against the target model, i.e., number of evasion attacks against the surrogate model that transfer;

2. transferability, i.e., the percentage of successful evasion attacks out of all the attempted evasion attacks against the target model.

Since the objective of the attack is to maximize the number of successful evasion attacks by using the smaller number of queries with the training purpose as possible, the first performance index is the most important for the attacker. Yet it's also important to take into account the second measure. Indeed, it firstly gives an indication of the influence of the size of $\mathcal{D}_{trn}$ on the similarity between the two models considered and trained on the specific dataset, since the greater the transferability, the better the alignment, keeping fixed the perturbation of the instances; secondly, this measure has been extensively studied in the literature [17, 33].
Finally, recall that AMEBA results to be an implementation of the Thompson sampling algorithm for solving the MAB problem, that incorporates the probabilistic sampling based on the Beta distribution in order to choose the best action at each turn. Moreover, the specific partitioning of D into the three datasets may introduce a bias in the results. Then the performance evaluation of the strategy cannot be based on a single configuration of the parameters of the algorithm, that consists in the three datasets, keeping the perturbation adopted by FGV fixed. Then the performance measures obtained by both the baseline and AMEBA are the average obtained in 10 different runs in which the partitions of D in the three datasets differs.

5.3 Experimental Results

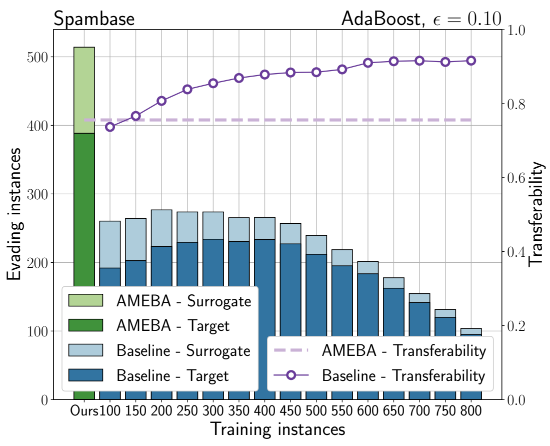
In this section, the experimental results are analysed and discussed. The experimental results of the tests in the binary and multiclass classification task setting are presented.

5.3.1 Binary classification task

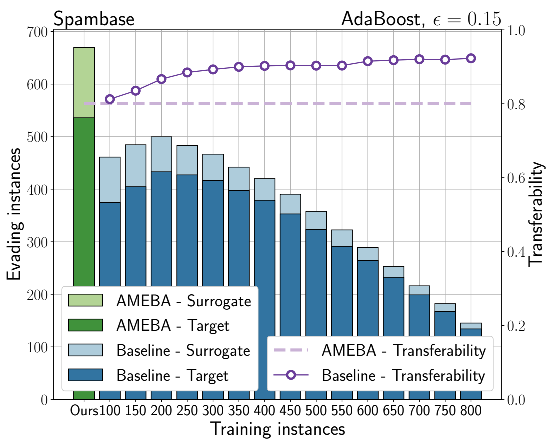
Spambase

In Figure 5.2 are shown the results on the Spambase dataset. The outer - light colored bars represent the number of evasion attacks successfully crafted against the surrogate model, while the inner bars show those which turned out to be effective also on the target model; the lines, instead, show the value of transferability. The results for Random Forest and AdaBoost are very similar and indicate that AMEBA outperforms the best-performing baseline. For example, in the case of Adaboost as target and perturbation $\epsilon = 0.1$, AMEBA allows to craft 390 effective adversarial example against the 234 produced using the best performing

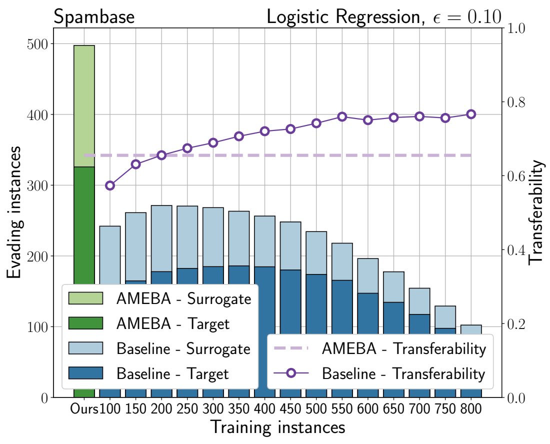
baseline, that corresponds to an increment of 66% in the number of successful evasion attacks. The gap between the number of successful evasion attacks obtained using the two strategies decreases as the perturbation increases at $\epsilon = 0.15$, since becomes easier to generate adversarial examples. However AMEBA continues to outperform the best performing baseline. Indeed the number of successful evasion attacks produced by the best baseline is 433 and the one obtained by AMEBA is 536 (+ 24%).
The results for logistic regression confirms the trend seen for the other two target models. More precisely, with perturbation $\epsilon = 0.10$, the best performing baseline generates only 186 successful evasion attacks, while AMEBA generates 326 successful evasion attacks (+75%). Moreover, when the perturbation is increased at $\epsilon = 0.15$, the best performing baseline generates 351 successful evasion attacks, while AMEBA generates 437 (+ 25%).
Looking at the transferability, it results evident that it improves as the number of queries spent for the surrogate model training increases. However, using a too large number of query instances for the training phase doesn't allow to exploit the high transferability obtained because of the small number of queries available for performing evasion attacks. The plots in 5.2 clearly show that the amount of queries that allows to maximize the number of successful evasion attacks could change a lot across different attack scenario, so it's hard to identify. The trade-off between the two phases of the attack is automatically handled by AMEBA, keeping the transferability very high even though not equal to the one obtained by the best performing baseline. For AdaBoost, the transferability obtained by AMEBA is 76% for $\epsilon = 0.1$ and 80% for $\epsilon = 0.15$, while from 85% to 87% when considering the baseline that produces the largest number of successful evasion attacks; if the target model is logistic regression, the transferability obtained by the baseline is 65% with both the perturbations, while 70% for the baseline.
The results for Logistic Regression seems counter-intuitive, since Linear SVM and logistic regression presents similarities, as shown in Section 2.2.4, but both the transferability and the number of evasion attacks against the logistic model are lower than the results obtained for the other two targets. However Demontis et. al. [17] shown throuhg their experiments that the similarity between the two model is not relevant in increasing the transferability.



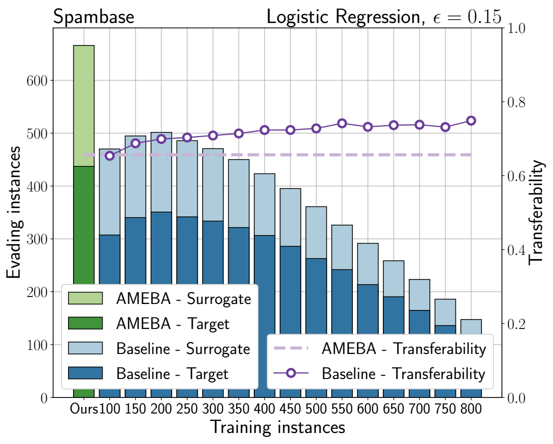
(a)



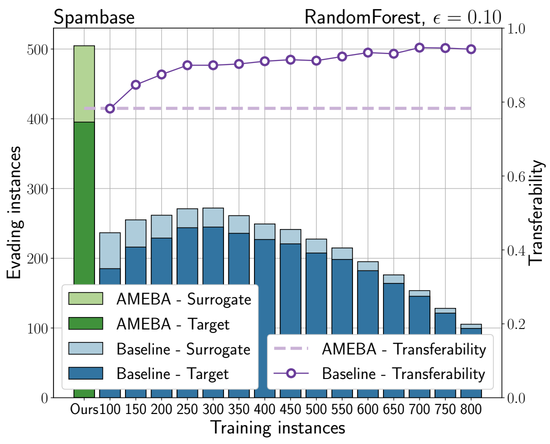
(b)



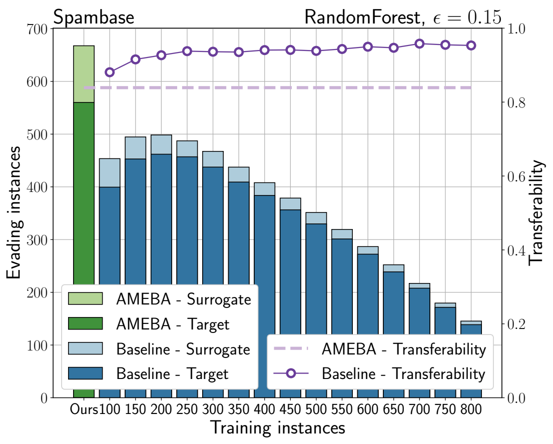
(c)



(d)



(e)



(f)

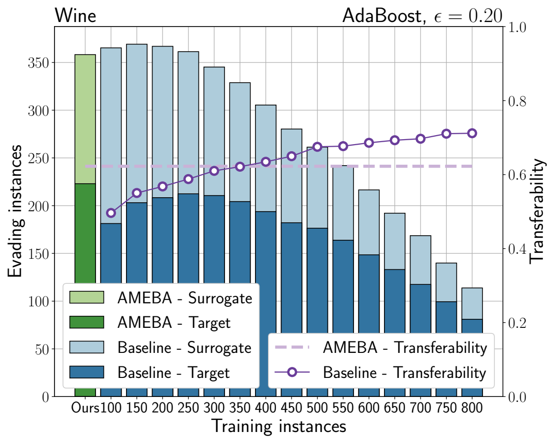
Figure 5.2: Experimental results on the Spambase dataset

Wine

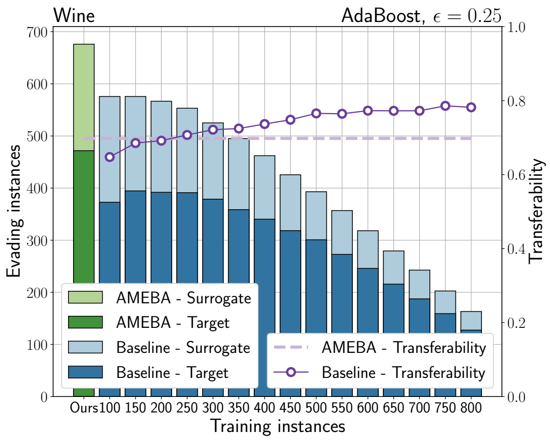
The results for the Wine dataset, shown in Figure 5.3, confirm that AMEBA surpass the baseline strategy in terms of number of successful evasion attacks. However, in this setting is more difficult to generate advexps than for the Spambase dataset. Then the results on the three target models are very close.
For $\epsilon = 0.20$, the most difficult model to attack is Adaboost, against which the best performing baseline generates 212 successful evasion attacks while AMEBA allows to generate 223 (+5%) and the transferability is near 60% for both the attackers. Logistic regression results to be the easiest model to attack with the Wine dataset, since the best performing baseline generates 250 successful evasion attacks while AMEBA allows to generate 263 (+5%) and the transferability is 67% for the best performing baseline and 64% for AMEBA. Even though the improvement in the number of successful evasion attacks is not as great as for Spambase, the result achieved by AMEBA is already positive, since the attacker doesn't know which is the best splitting of the queries.
When the perturbation increases to $\epsilon = 0.25$, evading the target model becomes easier and the difference in the number of generated advexps between the two attackers becomes apparent. For example, for Adaboost the best performing baseline produces 395 successful evasion attacks while AMEBA 471 (+20%) and the transferability is near 70% for both the attackers. For logistic regression the gap is smaller in terms of number of successful advexps, since the best performing baseline produces 416 successful evasion attacks while AMEBA 478 (+15%). However the gap in transferability is greater, since the transferability is 75% for the best performing baseline and 70% for the AMEBA.

CodRNA

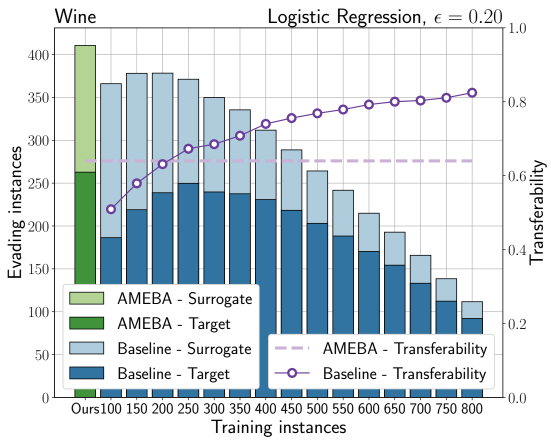
Finally, we report in Figure 5.4 the results for the CodRNA dataset. The results are very positive, also better that the ones obtained with the other datasets. Indeed AMEBA does not just generate a higher number of successful evasion attacks than the best performing baseline, but also improves the transferability over most baselines.
The results across the three different target models are very similar, so we take as reference AdaBoost. With perturbation $\varepsilon = 0.10$, the best-performing baseline produces 287 successful evasion attacks, while AMEBA can craft 419 successful attacks (+46%). The evasion attacks crafted by AMEBA have a transferability of 66%, which is very close to the transferability of the baseline producing the largest number of successful attacks (67%). When moving to $\varepsilon = 0.15$, the best-performing baseline identifies 468 successful evasion attacks, while AMEBA produces 572 successful attacks (+22%). The evasion attacks crafted by AMEBA have a transferability of 77%, a value which improves over most baselines and is close to the transferability of the baseline producing the largest number of successful attacks (77%).



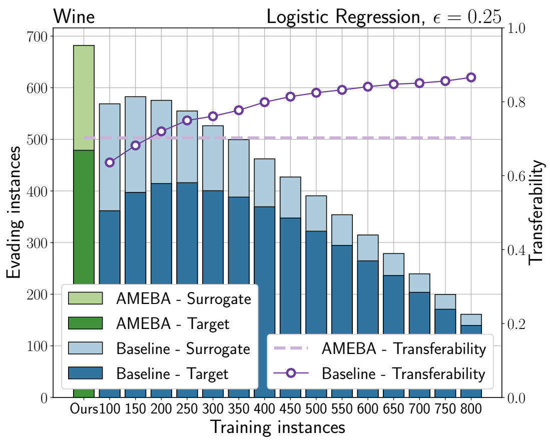
(a)



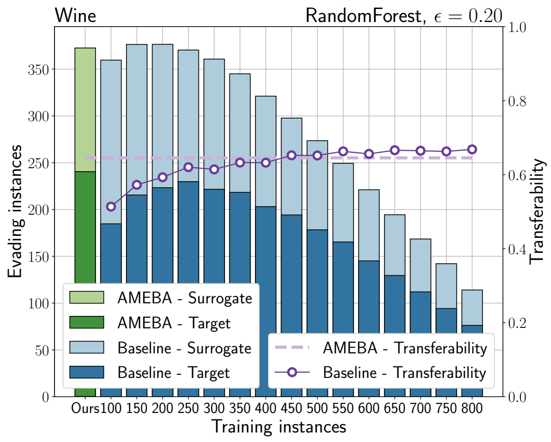
(b)



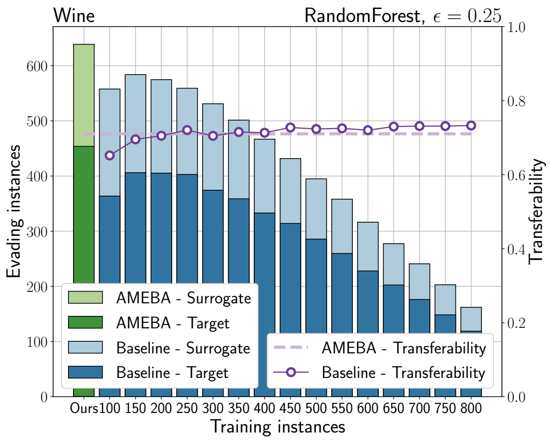
(c)



(d)

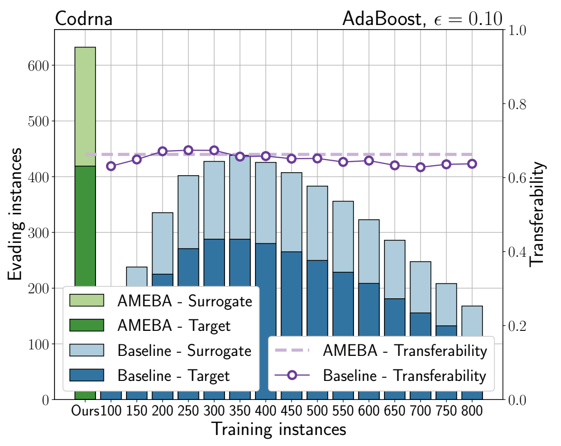


(e)

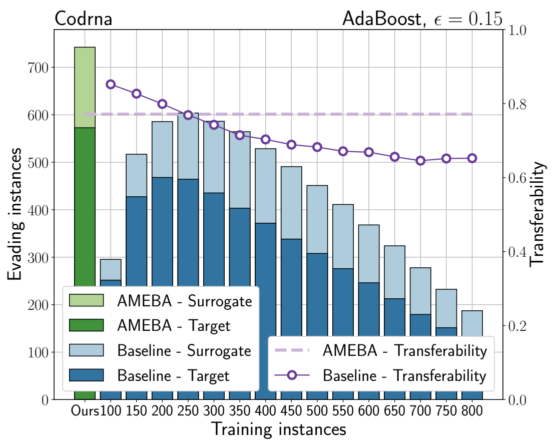


(f)

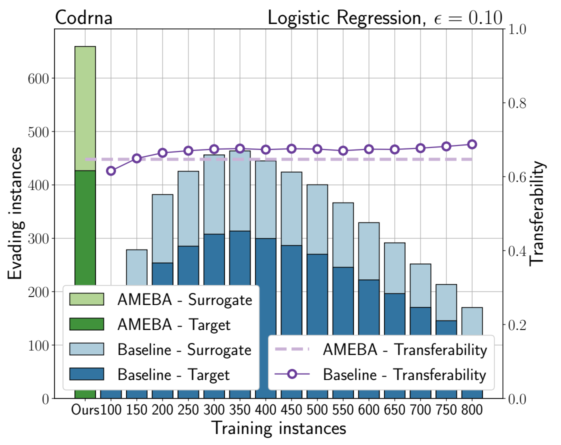
Figure 5.3: Experimental results on the Wine Quality dataset



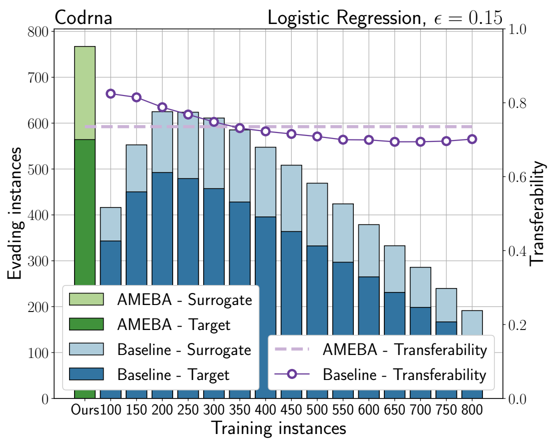
(a)



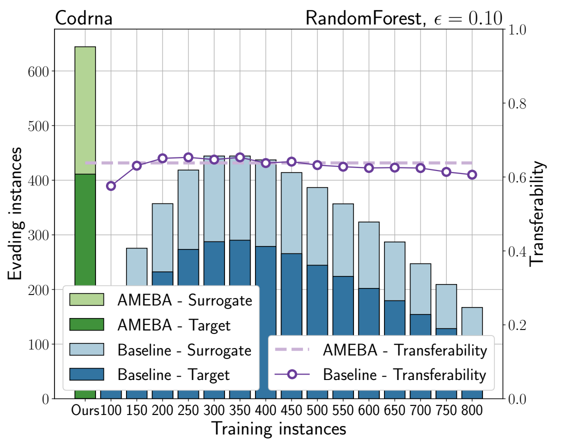
(b)



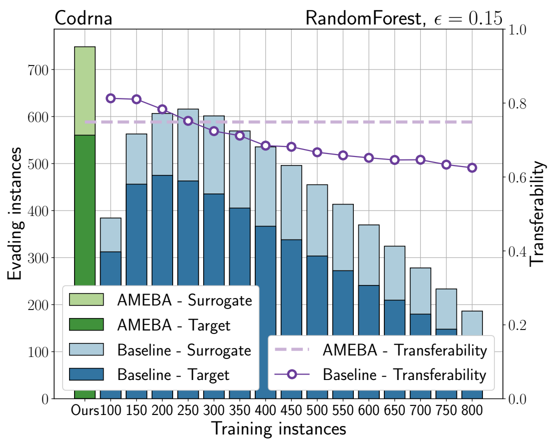
(c)



(d)



(e)

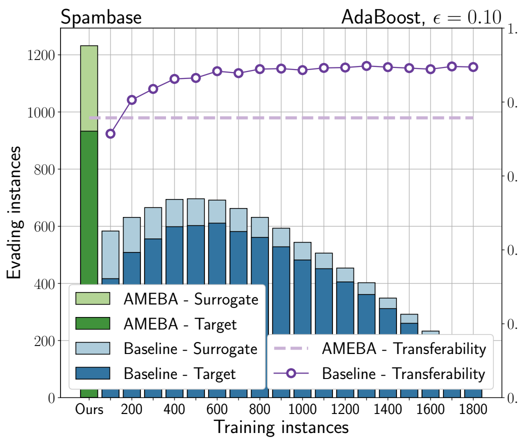


(f)

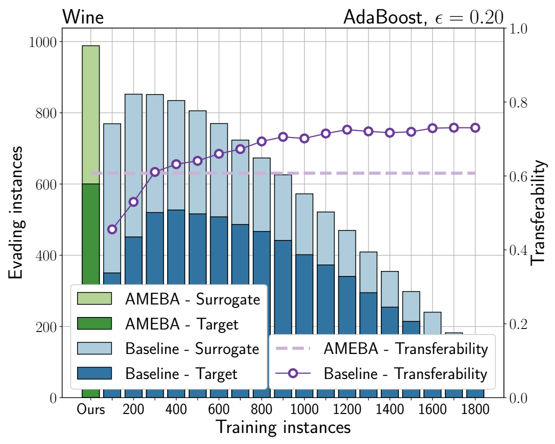
Figure 5.4: Experimental results on the CodRNA dataset

5.3.2 The Impact of Higher Number of Queries

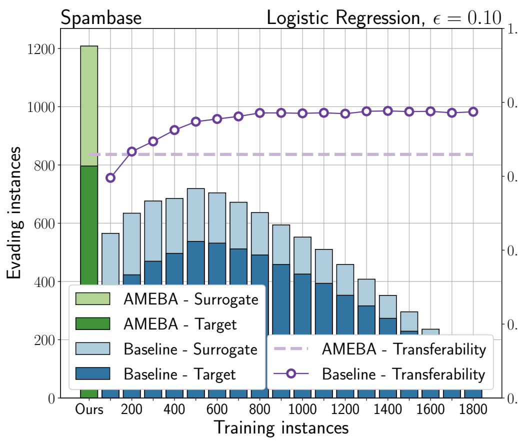
In order to show that the experiments are not biased by the chosen number of queries, which is an assumption on the attacker's power, in Figure 5.6 are reported additional experimental results under the assumption that the attacker can perform $T = 2000$ queries to the target model, rather than just $T = 1000$, with the smallest maximum perturbation ϵ for each dataset. The results shown in the figures are similar to the ones shown with in the plots in Section 5.3.1. We observe that increasing the number of queries used for training typically provides a better transferability for the baseline: most cases show a monotonic increase in transferability. However, a better transferability does not necessarily lead to a larger number of successful evasion attacks: the best-performing baseline in terms of successful evasion attacks typically uses a relatively low number of queries in the training phase, so that more evasion attempts are possible. Our experiments clearly show that the amount of successful evasion attacks generated by AMEBA still outperforms the best-performing baseline, for all datasets and models. The results obtained in the case of RandomForest and AdaBoost are quietly similar for all the datasets. In the case of the Random Forest model trained over the Spambase dataset, the best-performing baseline crafts 644 successful evasion attacks, while AMEBA can produce 902 successful attacks (+40%). With target logistic regression, the results are slighly different, since AMEBA succeeds to produce only 800 successful evasion attacks, result different than the one obtained with budget $T = 1000$ (cf figure 5.2), where against LogisticRegression AMEBA produces more successful advexps than against RandomForest and AdaBoost. This result shows that a bigger budget could lead to a different results, but AMEBA still outperforms the best performing baseline.
At the same time, the transferability of the evasion attacks produced by AMEBA stays in a very acceptable range, from 60% at worst to 78% at best across the different settings. An interesting point comes from observing the results for target LogisticRegression with dataset Wine and CodRNA. Following the trend of the experiments with budget $T = 1000$, the transferability from LinearSVM to LogisticRegression on Wine increases as the surrogate training set size increases. In the case of CodRNA dataset, with budget $T = 1000$ the transferability between the two models involved doesn't improve as the size of the training set increases, while with budget $T = 2000$ the transferability starts to increase after size 1000, a countertrend result compared with those obtained in case of RandomForest and AdaBoost as target. This is probably due to the inherent similarity between the Logistic Regression model and LinearSVM, as pointed out in section 2.2.4.


Spambase
AdaBoost, $\epsilon = 0.10$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

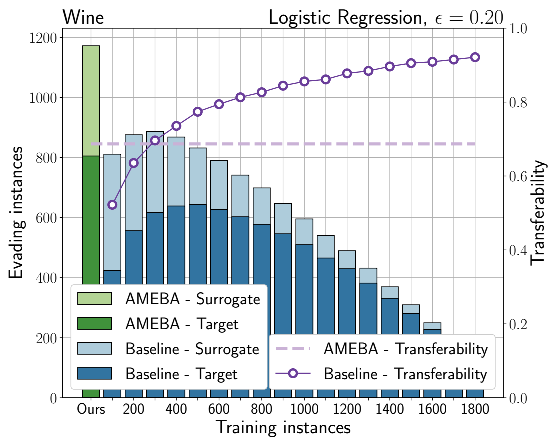
(a)


Wine
AdaBoost, $\epsilon = 0.20$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

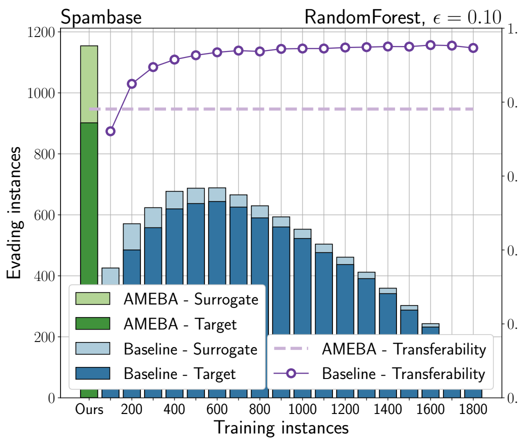
(b)


Spambase
Logistic Regression, $\epsilon = 0.10$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

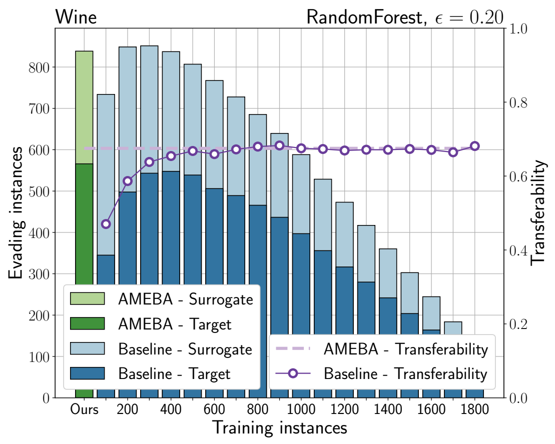
(c)


Wine
Logistic Regression, $\epsilon = 0.20$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

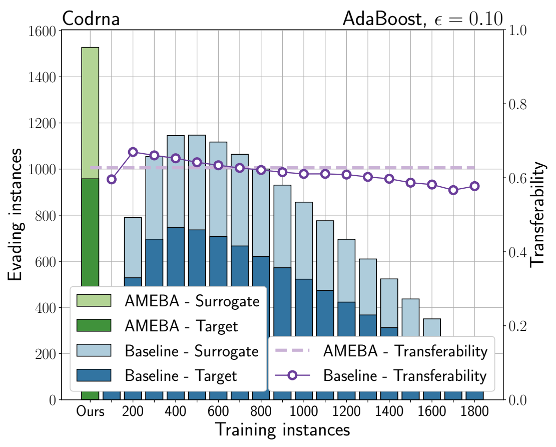
(d)


Spambase
RandomForest, $\epsilon = 0.10$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

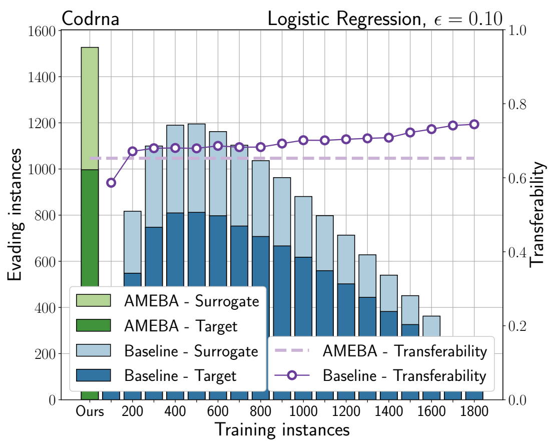
(e)


Wine
RandomForest, $\epsilon = 0.20$
Evading instances
Transferability
Training instances
AMEBA - Surrogate
AMEBA - Target
Baseline - Surrogate
Baseline - Target
AMEBA - Transferability
Baseline - Transferability

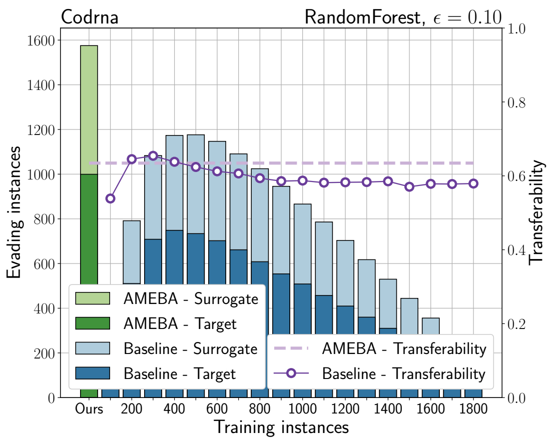
(f)



(a)



(b)

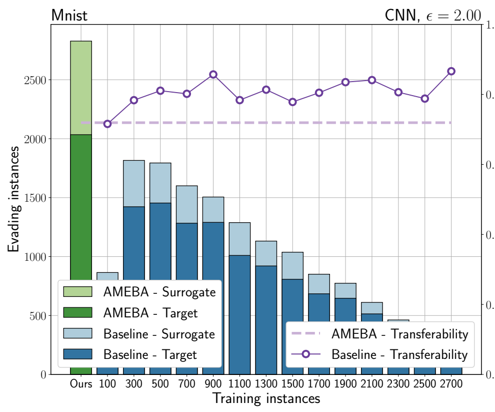


(c)

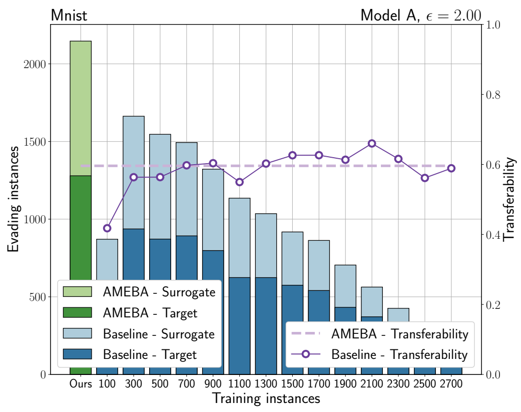
Figure 5.6: Experimental results for $T = 2000$ queries

5.3.3 Multiclass Image Classification Task

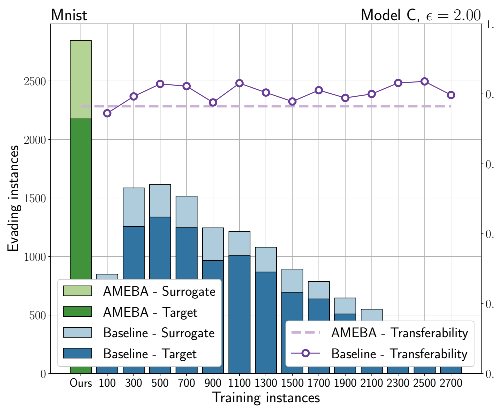
The results about the application of AMEBA in the context of a image classification task with perturbation $\epsilon = 2.0$ are shown in Figure 5.7. AMEBA outperforms the best performing baseline in all the attack scenario. For example, in the case of MODELC, the most vulnerable model, the best baseline produces only 1337 successful evasion attacks while AMEBA produces 2176 effective advexps (+62%) and the transferability is not very distant, 77% for AMEBA and 82% for the best performing baseline. For MODELA, that's the toughest model to attack in this setting, the transferability is very low, near 50%, for both the attackers, but AMEBA still outperforms the best performing baseline in terms of the number of successful advexps, 1111 vs 927 (+20%).
In figure 5.8 are reported the results with $\epsilon = 3.0$. As expected, the number of successful evasion attacks increases both for AMEBA and the baseline, but AMEBA still improves the baseline results. For example, in the case of MODELC, the best performing baseline succeeds to craft 1864 attacks and AMEBA 2402 attacks (+28%), while for MODELA without the dropout layer the best performing baseline attacker succeeds to craft 1517 attacks and AMEBA 1852 attacks (+22%). In general, the percentage increasing in the number of successful evasion attacks against the targets ranges from +22% to +40%. The transferability of the evasion attacks crafted by AMEBA is closely the same as the one achieved by the best perfoming baseline or better, however it ranges from 70% against MODELA to 87% against MODELC.
There are two interesting observations to point out. Firstly, the transferability doesn't show the same increasing trend as seen for Spambase and Wine, probably because the size of the architectures, the use data augmentation and early stopping are three factors that reduce the impact of small $\mathcal{D}_{trn}$ on the network generalization capabilities. Indeed the augmentation applied on $\mathcal{D}_{trn}$ allows to show the network a lot of different instances also starting from a small pool, while a small deep architecture doesn't need a very large training set compared to deeper architectures. Moreover, the use of early stopping allows to prevent overfitting. The second observation is that the results show surprisingly that MODELA is easier to evade than its variant without drop-out layers. Our conjecture about this might come from the observation that random noise is not necessarily a good approximation of adversarial noise [6].



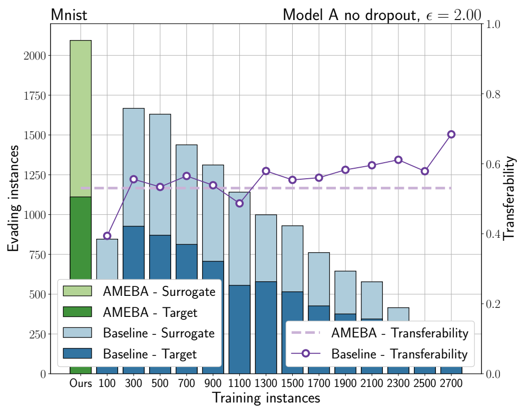
(a)



(b)

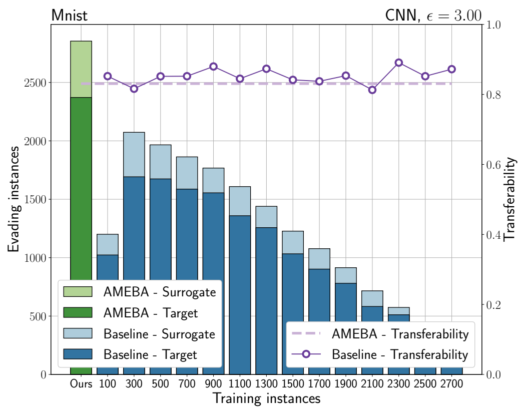


(c)

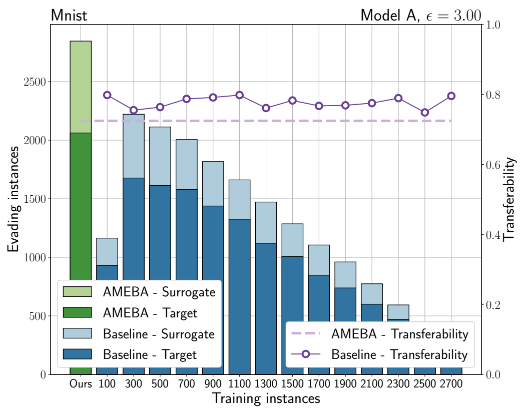


(d)

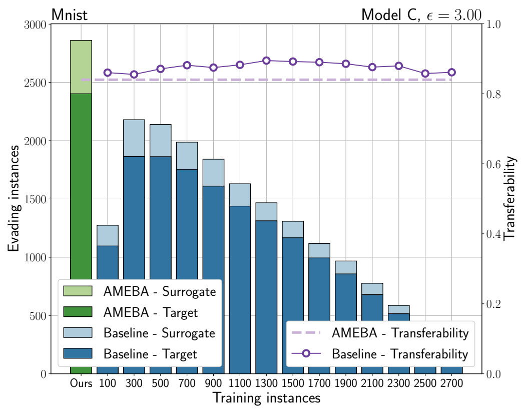
Figure 5.7: Experimental results on the MNIST dataset for $T = 3000$ and $\epsilon = 2.0$



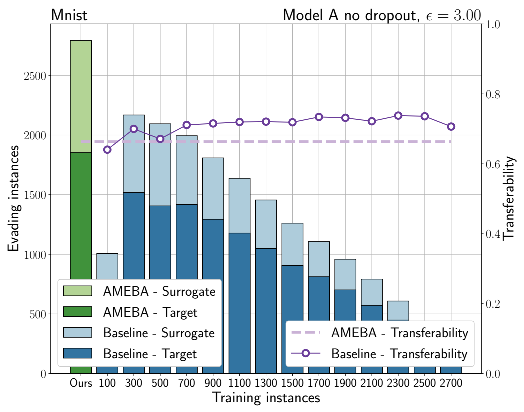
(a)



(b)



(c)



(d)

Figure 5.8: Experimental results on the MNIST dataset for $T = 3000$ and $\epsilon = 3.0$

Spambase			**Wine Quality**		
$\varepsilon = 0.10$	**Total Time**	**Average**	$\varepsilon = 0.20$	**Total Time**	**Average**
Random Forest	607	1.53	Random Forest	392	1.63
AdaBoost	317	0.81	AdaBoost	214	0.96
Log. Regression	243	0.75	Log. Regression	241	0.92
$\varepsilon = 0.15$	**Total Time**	**Average**	$\varepsilon = 0.25$	**Total Time**	**Average**
Random Forest	267	0.48	Random Forest	191	0.42
AdaBoost	150	0.28	AdaBoost	176	0.37
Log. Regression	99	0.23	Log. Regression	143	0.30

CodRNA		
$\varepsilon = 0.10$	**Total Time**	**Average**
Random Forest	312	0.76
AdaBoost	193	0.46
Log. Regression	103	0.24
$\varepsilon = 0.15$	**Total Time**	**Average**
Random Forest	261	0.46
AdaBoost	133	0.23
Log. Regression	61	0.11

Table 5.5: Performance evaluation of AMEBA using Linear SVM as surrogate model. The Total Time column reports the total running time of AMEBA, while the Average column reports the average time to perform a successful evasion attack. Times are expressed in seconds.

5.4 Performance Evaluation

An interesting point to be investigated is the time needed to run an attack using AMEBA. Indeed the running time required by AMEBA could be larger than the one needed for the traditional two step attack strategies proposed in prior work, most notably because the number of *Train* and *Attack* actions is dynamically chosen, hence the surrogate model can be trained many times, i.e., every time the *Train* action is performed. The analysis is presented for both the experimental settings considered. The performance metrics used are the total time spent to carry out an adaptive attack through AMEBA (up to query budget exhaustion) and the average time spent to craft a successful evasion attack. Each result is the average over 10 different runs of AMEBA on a standard commercial machine over the different attack settings considered.

Binary Classification Task

Our experiments in the binary classification task setting show that AMEBA is perfectly appropriate for pratical usage. This is due to the efficiency of both the attack strategy and the supervised learning algorithms for simple models like Linear SVM.
The results about our experiments are shown in Table 5.5. With the lower perturbation between the two used for each dataset, the total time required to complete the attack on a budget of queries is in general higher than the times obtained with the higher perturbation. This is due to the higher number of *Train* actions that are performed in the setting with the lower perturbation and the number of times in which the entire $\mathcal{D}_{atk}$ is examined without finding a usable base instance. Indeed the lower the perturbation, the more difficult is to craft valid evasion attacks and the higher will be the number of *Train* actions performed. Moreover, the running

MNIST		
$\varepsilon = 2.0$	**Total Time**	**Average**
CNN	1658	0.81
MODELC	1363	0.62
MODELA	17328	13.54
MODELAdropless	18676	16.81
$\varepsilon = 3.0$	**Total Time**	**Average**
CNN	1107	0.47
MODELC	938	0.39
MODELA	1097	0.53
MODELAdropless	1528	0.82

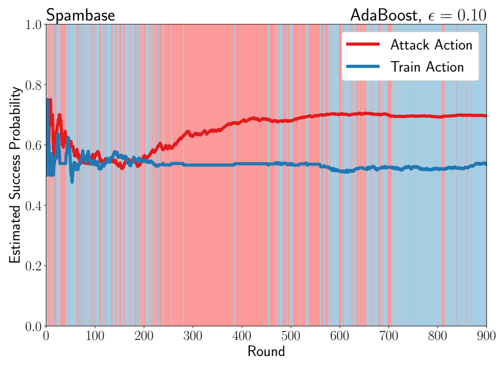
Table 5.6: Performance evaluation of AMEBA using LeNet as surrogate model. The Total Time column reports the total running time of AMEBA, while the Average column reports the average time to perform a successful evasion attack. Times are expressed in seconds.

time is also influenced by the classification time needed by the target model, that's higher for the decision tree model. In the worst case, so the setting with Random Forest, perturbation $\epsilon = 0.1$ and the Spambase dataset, AMEBA carries out 10 minutes (607 seconds) to perform an entire attack that exploits 1000 queries, while only around 4 minutes if the perturbation is $\epsilon = 0.15$. Also the average time reflect this difference, since it switch from 1.53 successful evasion attack per second to 0.48. In the best case, with Logistic Regression and the CodRNA dataset, the the attacker uses around 1 and half minutes (103 seconds) to perform an entire attack that exploit 1000 queries, while only around 1 minute if the perturbation is $\epsilon = 0.15$.
In general, over the three datasets, AMEBA requires more time against Random-Forest than the other two target model, but the time required is always reasonably low. Also the average time required to craft a successful evasion attack is very low, since it's under 2 seconds in the worst case and less than 1 second in most cases. These results confirm that the Thompson sampling algorithm is an effective choice to implement the adaptive attack strategy.

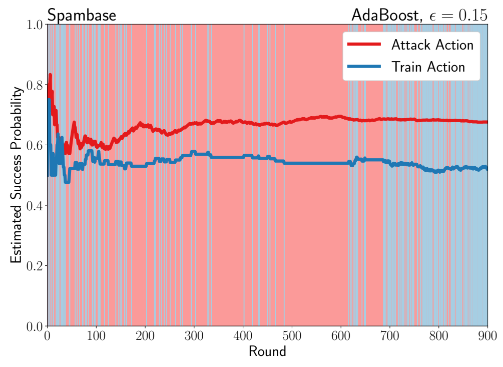
Multiclass Classification Task

When the surrogate and target models are CNNs, the time required to carry out an adaptive black-box attack through AMEBA increases with respect the results obtained in the previous setting, but it remains appropriate for practical usage.
In table 5.6 the time results are presented for each different target model on the dataset MNIST, with budget $T = 3000$ and the two perturbations. The different budget makes not possible to compare the total times with the ones obtained in the previous setting, but the average time to craft a successful evasion attack can be compared. When the evasion attacks transfer well, as in the case for MODELC and CNN, the average times to perform a successful evasion attack through AMEBA are aligned to the ones obtained in the previous setting against simpler targets (Section 5.4). When the evasion is more challenging, the average time required increases. By observing the results with perturbation $\epsilon = 2$, it's evident the impact of the training steps in the total and average time. For example, in the case of MODELA, more training steps are performed, since the surrogate model needs to be refined in order to craft valid evasion attacks. Moreover many instances

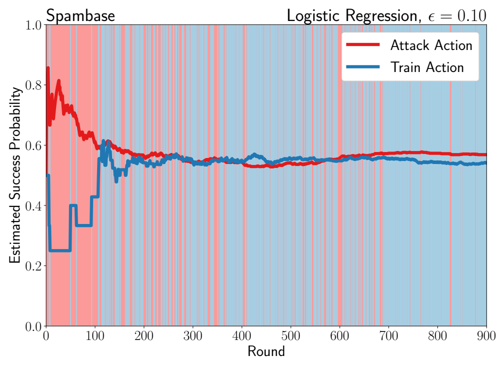
don't evade the surrogate model because of the low perturbation. Then, against MODELA, AMEBA requires around 5 hours to complete the evasion attack using the entire budget, while only 27 minutes are required for the attack against CNN. When the perturbation raises to $\epsilon = 3$, also the total time required for completing the attack against MODELA drops, indeed it requires only 18 minutes. The evasion is made easier thanks to the higher perturbation, so less training steps are needed to be performed than in the setting with $\epsilon = 2$.



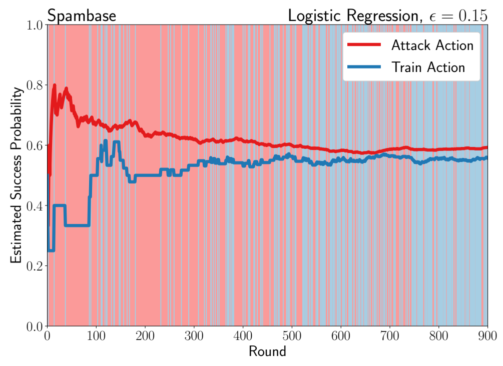
(a)



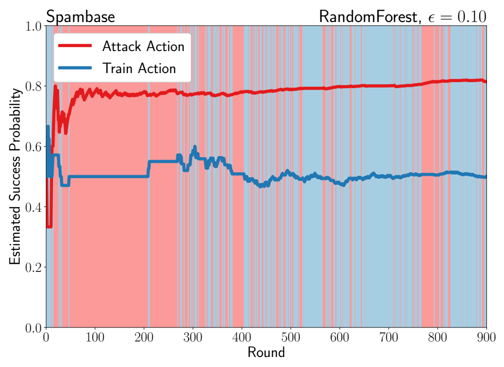
(b)



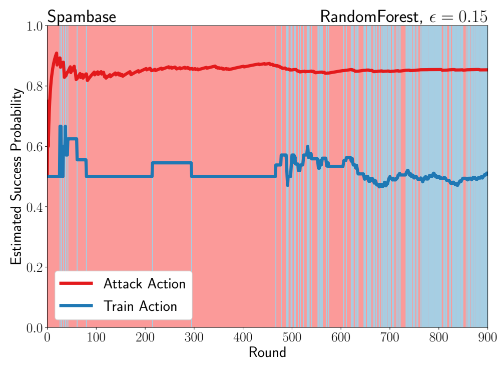
(c)



(d)



(e)



(f)

Figure 5.9: Actions and mean rewards on the Spambase dataset

5.5 Discussion

5.5.1 Why AMEBA Works?

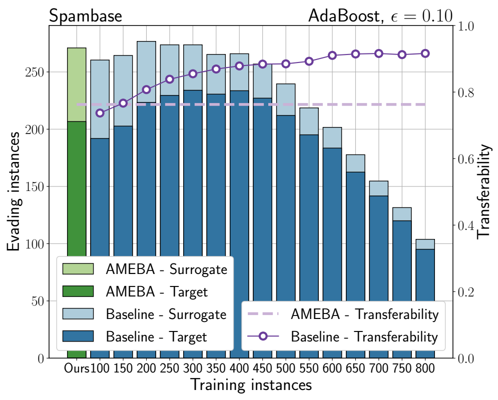
In order to understand why AMEBA works, it is necessary to study the impact of two key characteristics of the attack strategy: the dynamic alternation between the two actions and the implementation of the $\mathcal{D}_{atk}$ set as a queue. The former is a consequence of the reduction of the problem to MAB, solved by using Thompson sampling, while the latter is a consideration about the attacker's behaviour in the threat model considered.

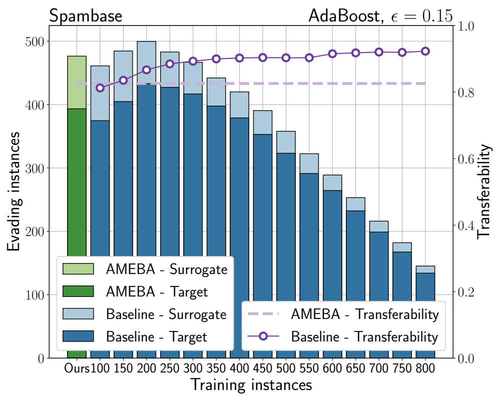
Dynamic Alternation between the Two Actions

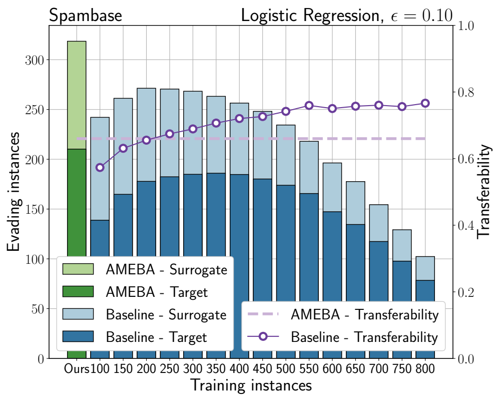
About the first key factor, in Figure 5.9 are shown 6 different runs of the AMEBA algorithm against the three target models on the Spambase dataset, using the two perturbations considered in Section 5.3.1. The figure uses red and blue lines to show the trend of the average reward for the *Attack* and the *Train* actions respectively; the background of the plots shows instead which of the two actions was taken at each round. When the perturbation is smaller, $\epsilon = 0.1$ and the target model is quite vulnerable to evasion attacks (the two decision tree ensemble), AMEBA shows an expected trend: it firstly performs a small number of train actions, then moves to *Attack* actions in order to exploit the good transferability between surrogate and target model and, at the end, *Train* actions become popular because of the impossibility of crafting adversarial examples with the remaining instances in $\mathcal{D}_{atk}$. Against logistic regression, that's more difficult to evade, the choices are different: firstly AMEBA uses the *Attack* action, but at a certain point encounters difficulties in the evasion. Then it recurs to the augmentation of $\mathcal{D}_{trn}$ through the *Train* action in order to improve the quality of the surrogate. Only when the mean reward of the *Attack* action turns to exceed that of the *Train* action, AMEBA switches back to evasion attack crafting. With perturbation $\epsilon = 0.15$, evading the target models becomes easier then AMEBA exploits more times the *Attack* action, as shown by (d), (e) and (f) in Figure 5.9.

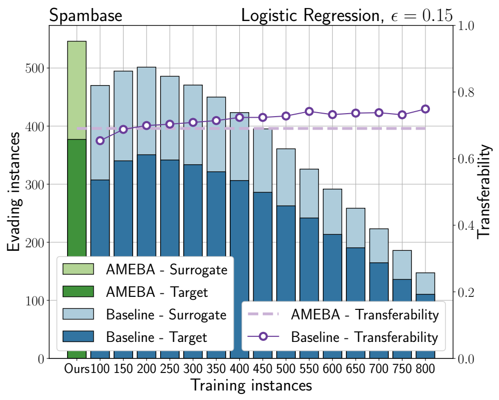
The Effect of organizing $\mathcal{D}_{atk}$ as Queue

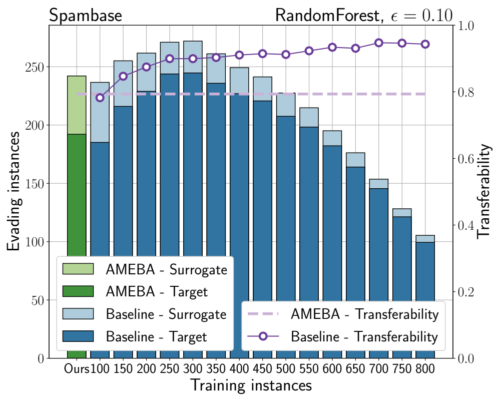
About the second key factor, Figure 5.10 shows the results obtained on the Spambase dataset if we did not organize $\mathcal{D}_{atk}$ as a queue and rather discarded instances which were used to generate an advexps that could not evade the surrogate model. In case of tree ensembles, the adaptive strategy doesn't suffice the improve significantly the number of successful evasion attacks with respect to the baseline starting from 100 instances, while it was used to outperform the baseline (Figure 5.2). For example, in the case of RandomForest and with perturbation $\epsilon = 0.10$, AMEBA produces 192 effective adversarial examples, while the best performing baseline 245. With perturbation $\epsilon = 0.15$, the gap improves, since AMEBA produces only 356 effective evasion attacks and the best performing baseline 427. Surprisingly, against logistic regression AMEBA allows to overcome the best performing baseline even if $\mathcal{D}_{atk}$ is not organized as a queue. Indeed, with perturbation $\epsilon = 0.10$, AMEBA allows to perform 210 successful adversarial attacks, while the best performing baseline only 186. If the perturbation increases to $\epsilon = 0.15$, AMEBA induces to craft 377 successful advexps, while the best performing baseline only 351. However, if we compare the results obtained for Logistic Regression with $\mathcal{D}_{atk}$ organized as queue or not, it's evident the advantage of organizing $\mathcal{D}_{atk}$ as queue. Indeed the intuition under the success of the organization as queue is that, since the surrogate model is refined over time, instances leading to a failure at a given round might lead to successful evasion attacks in a later round. This confirms that adaptive attack strategies have an inherent potential to outperform traditional two-step attack strategies, where a large number of instances can never be used to craft successful evasion attacks.


(a)


(b)


(c)


(d)


(e)

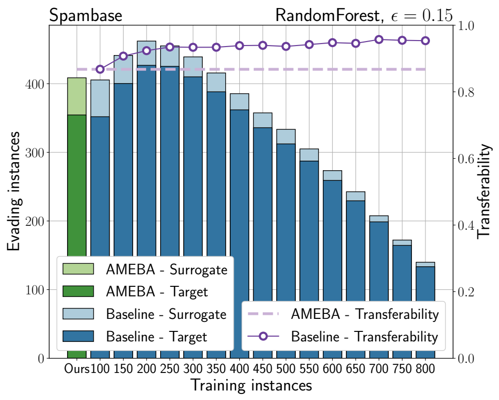

(f)

Figure 5.10: Experimental results on the Spambase dataset (without queue)

5.5.2 The Importance of the Adaptive Attack Strategy

The experimental evaluation points out two important observations which motivate the importance of designing adaptive attack strategies like AMEBA.
Firstly, finding the optimal trade-off between surrogate model training and evasion attack crafting is far from straightforward. The effectiveness of the black-box evasion attack run adopting the static two-step strategy, evaluated using the number of successful advexps against the target, shows the shape of a bell-curve as the number of queries spent to initially train the surrogate model increases. The size of the training set that allows to obtain the maximum number of successful evasion attacks may be very difficult to predict (see Figure 5.2 and Figure 5.3), then the attacker cannot know in advance the right time to switch from surrogate model training to evasion attack crafting.
Secondly, the two-step adaptive strategy proposed, AMEBA, do not just approximate the best-performing baseline, but in certain scenarios can outperform it, as we observed on the Spambase or MNIST dataset. The first key point here is that the adaptive two step strategy allows to automatically choice when it's better to spend query to improve the quality of surrogate model. The second key point consists in giving other chances to instances in $\mathcal{D}_{atk}$ that, in previous iterations, could not be used to craft successful evasion attacks. The static two step attack strategy expects to discard them, since it isn't possible to improve the quality of the surrogate model after the initial training phase. In the context of the adaptive strategy is more convenient to store them and try newly the attack later, since the model is likely to be refined soon. The adaptive alternation between the two actions allows to enable attacks that previously were not possible, because of the changes in the decision boundaries established by the surrogate model after training on the augmented $\mathcal{D}_{trn}$. The reuse of previously misclassified or unusable instance impacts a lot the effectiveness of the strategy, fact that can be appreciated by comparing, for example, the results in Figure 5.2 against those in Figure 5.10.
Finally, AMEBA is a novel attack but also an optimization of traditional attack strategies based on transferability. The attack strategy is a way to decide how to choose an action depending on the setting and it's only one of the factors that determine the success of the overall attack. Another fundamental factor is the white-box attack chosen to craft the adversarial examples. Then the effectiveness of AMEBA in certain settings depends on the attack used: if the adversarial examples crafted show really bad transferability because of the white-box attack chosen, then many *Train* actions will be performed consuming the entire budget. AMEBA does not change the landscape of the research on the defenses against adversarial examples transferability, that's still an open problem [47], but it can be adopted to make the security evaluation of machine learning models more meaningful in practice.

Chapter 6

Conclusion

Adversarial machine learning has increasingly gained popularity in the last decade, due to the wider adoption of machine learning in very heterogeneous fields. The security and robustness of machine learning models are essential mostly in security-oriented applications, like anomaly based intrusion detection systems or fraud detectors. Although many attack types exist, in this work we focused on evasion attacks. As the word ”evasion” suggests, the objective is to induce the target model to misclassify a carefully crafted instance $\vec{z}$, called adversarial example, which is very similar to the original one $\vec{x} \in \mathcal{X}$.
In this thesis we discussed the problem of maximizing the number of successful evasion attacks in the black-box setting. In particular, the details of the target model are hidden and only the label predicted for a given query is returned. Moreover, a typical constraint in the black-box settings concern the budget of the attacker, i.e., the maximum number of queries to the target model. A traditional and practical strategy to perform black-box evasion attacks in the black-box setting is the two-steps attack strategy presented by Papernot et. al. [34]. It's articulated in two steps: (i) query the target model to collect predictions for the surrogate training set and train the surrogate model that has to approximate the target, (ii) craft evasion attacks against the surrogate model and try to transfer them to the target. Given the constraint about the maximum number of queries, an inherent tension emerges between the two steps of the attack strategy. Indeed the attacker would like to obtain the best approximation of the target model and the highest number of successful evasion attacks as possible, but improving the surrogate model needs to retrieve predictions from the target model, i.e., spending queries for the training step. The traditional two-step attack strategy is suboptimal, since it assumes the two steps as strictly separated and requires to define before the attack execution the number of queries spent for the first step to perform the attack. It doesn't allow the attacker to exploit all the possible actions available, such as dynamically intertwining the two steps in order to adapt the number of queries used for each step during the execution of the attack.
The solution proposed in this work is AMEBA, the first adaptive approach to the black-box evasion of ML models. The strategy allows to learn the best alternation of the two actions for surrogate model training and evasion attack crafting during the execution of the attack. AMEBA exploits the formal reduction from the two-step evasion attack problem to the Bernoulli MAB problem that allows to implement AMEBA using one of the solving algorithms for the MAB problem. In this work, we used the Thompson sampling algorithm. For each possible query of the attacker's budget B, the attacker exploits a Beta prior to adaptively choose

whether to use the query to retrieve a prediction and improve the target model or to submit an advexp. We experimentally showed that AMEBA can outperform the traditional two-step attack strategy at least in terms of number of successful evasion attack crafted. In particular, we tested AMEBA in the context of binary classification tasks and an image classification task with a variety of supervised learning algorithms, like deep neural networks. Moreover, the time required to AMEBA to perform a complete attack with budget $T = 1000$ is at least 10 minutes, so it's perfectly applicable in practice. The success of AMEBA is mainly due to two key points: the ability of learning the most convenient action to take, thanks to the good heuristic used, and the choice of not discarding instances that a certain point cannot be used to craft an advexp.

To the best of our knowledge, our work is the first to propose an adaptive two-step attack strategy to perform evasion attacks in the black-box setting exploiting transferability. In literature there are plenty of proposed attack algorithms and strategies, as described in Chapter 3. The practical black-box evasion attack against ML proposed by Papernot et. al. [34] achieved an important success and influenced other notable work in the area [33, 32]. However, none of this paper examines the inherent trade-off between the two steps during the attack and propose a way to switch from one to the other in an optimal way. The other prominent category of black-box attacks, the query-based, includes attacks that perform multiple queries to the target model in order to estimate its gradient [13]. Although recent papers focused on how to reduce the number of queries required by the query-based attacks [25, 31, 24], they continue to require a number of queries that's still order of magnitudes higher than the one obtained by transfer-based approaches. The two strategies are complementary: transfer-based attacks presents a relatively low success rate, although they are extremely efficient since they require only to query the target model to perform an evasion attack; query-based attacks guarantee a nearly perfect success rate, but they require a lot of more queries to evade a single instance. Moreover, the same surrogate model can be used to attack different targets (as non-differentiable models) and can be used to attack all the instances of interests, which makes the attack efficient and harder to be detected. For these reasons, the research community continues to have still interest in transferability. The complementarity guided prior work that proposed hybrid strategies that combine the two strategy in order to take the best from the two worlds [44, 15, 28]. Since hybrid attacks exploits concepts from the transfer-based attacks, they can take advantage from the adaptive approach proposed by AMEBA.

We highlight that the attack strategy proposed is not focused on the evasion attack crafting, but in how to deal with the inherent tension between surrogate model training and evasion attack crafting in a query-limited setting. We expect our results to immediately generalize to other evasion attacking crafting algorithm besides the FGV algorithm which we used, like FGS, CW [11] and any other new attack algorithm designed to evade specific defense techniques like adversarial training [47].

We foresee several avenues for future work. In the immediate feature, we will attempt the reduction of the two-step evasion attack problem to other suitable Reinforcement Learning problems, so we will exploit other solving algorithms like Q-learning. Also, we would like to test different rewards for the *Train* action in our reduction to MAB, since the cross-validation score is just one of different plausible measures to explore. Secondly, other specific attacks to generate adversarial examples may be tested, like CW [11] or iterative methods [18]. Then, we would like to extend our experimental evaluation to more sophisticated attack strategies, where instances for surrogate model training are not randomly chosen, but rather crafted to maximize the similarity between the surrogate and the target [34]. Moreover, we plan to generalize our approach to the case where the output of the target model is not just a class label, but rather a confidence score or a probability vector. This additional amount of information might support the design of more sophisticated heuristics to assign the rewards of the two actions in our reduction to MAB.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Logistic function wikipedia. https://en.wikipedia.org/wiki/Logistic_function. Accessed: 2021-06-28.

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning.* Springer, 2018. ISBN 978-3-319-94462-3. doi: 10.1007/978-3-319-94463-0.

[3] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. D. Tygar. Can machine learning be secure? In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Paul Lin, Shiuhpyng Shieh, and Sushil Jajodia, editors, *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ASIACCS 2006, Taipei, Taiwan, March 21-24, 2006*, pages 16–25. ACM, 2006. doi: 10.1145/1128817.1128824. URL https://doi.org/10.1145/1128817.1128824.

[4] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Mach. Learn.*, 81(2):121–148, 2010. doi: 10.1007/s10994-010-5188-5. URL https://doi.org/10.1007/s10994-010-5188-5.

[5] Arjun Nitin Bhagoji, Warren He, Bo Li, and Dawn Song. Practical black-box attacks on deep neural networks using efficient query mechanisms. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XII*, volume 11216 of *Lecture Notes in Computer Science*, pages 158–174. Springer, 2018. doi: 10.1007/978-3-030-01258-8_10. URL https://doi.org/10.1007/978-3-030-01258-8_10.

[6] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *CoRR*, abs/1712.03141, 2017. URL http://arxiv.org/abs/1712.03141.

[7] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný, editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III*, volume 8190 of *Lecture Notes in Computer Science*, pages 387–402. Springer, 2013. doi: 10.1007/978-3-642-40994-3_25. URL https://doi.org/10.1007/978-3-642-40994-3_25.

[8] Cristopher Bishop. *Pattern recognition and machine learning.* Springer, 2006.

[9] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, 2018. URL https://openreview.net/forum?id=SyZI0GWCZ.

[10] Stefano Calzavara, Lorenzo Cazzaro, and Claudio Lucchese. AMEBA: an adaptive approach to the black-box evasion of machine learning models. In Jiannong Cao, Man Ho Au, Zhiqiang Lin, and Moti Yung, editors, *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, pages 292–306. ACM, 2021. doi: 10.1145/3433210.3453114. URL https://doi.org/10.1145/3433210.3453114.

[11] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 39–57. IEEE Computer Society, 2017. doi: 10.1109/SP.2017.49. URL `https://doi.org/10.1109/SP.2017.49`.

[12] Hongge Chen, Huan Zhang, Si Si, Yang Li, Duane S. Boning, and Cho-Jui Hsieh. Robustness verification of tree-based models. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12317–12328, 2019. URL `http://papers.nips.cc/paper/9399-robustness-verification-of-tree-based-models`.

[13] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In Bhavani M. Thuraisingham, Battista Biggio, David Mandell Freeman, Brad Miller, and Arunesh Sinha, editors, *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*, pages 15–26. ACM, 2017. doi: 10.1145/3128572.3140448. URL `https://doi.org/10.1145/3128572.3140448`.

[14] Minhao Cheng, Thong Le, Pin-Yu Chen, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=rJlk6iRqKX`.

[15] Shuyu Cheng, Yinpeng Dong, Tianyu Pang, Hang Su, and Jun Zhu. Improving black-box adversarial attacks with a transfer-based prior. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 10932–10942, 2019. URL `https://proceedings.neurips.cc/paper/2019/hash/32508f53f24c46f685870a075eaaa29c-Abstract.html`.

[16] Nilesh N. Dalvi, Pedro M. Domingos, Mausam, Sumit K. Sanghai, and Deepak Verma. Adversarial classification. In Won Kim, Ron Kohavi, Johannes Gehrke, and William DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 99–108. ACM, 2004. doi: 10.1145/1014052.1014066. URL `https://doi.org/10.1145/1014052.1014066`.

[17] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. pages 321–338, 2019. URL `https://www.usenix.org/conference/usenixsecurity19/presentation/demontis`.

[18] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9185–9193. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00957. URL `http://openaccess.thecvf.com/content_cvpr_2018/html/Dong_Boosting_Adversarial_Attacks_CVPR_2018_paper.html`.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6572`.

[21] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining Concepts and Techniques (Third edition)*. Elsevier, 2012.

[22] Mahdi Hashemi. Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation. *Journal of Big Data*, 6, 11 2019. doi: 10.1186/s40537-019-0263-7.

[23] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I. P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In Yan Chen, Alvaro A. Cárdenas, Rachel Greenstadt, and Benjamin I. P. Rubinstein, editors, *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011*, pages 43–58. ACM, 2011. doi: 10.1145/2046684.2046692. URL `https://doi.org/10.1145/2046684.2046692`.

[24] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 2142–2151. PMLR, 2018. URL `http://proceedings.mlr.press/v80/ilyas18a.html`.

[25] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL `https://openreview.net/forum?id=BkMiWhR5K7`.

[26] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2016.

[27] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition*. Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009. ISBN 9780135041963. URL `https://www.worldcat.org/oclc/315913020`.

[28] Mika Juuti, Buse Gul Atli, and N. Asokan. Making targeted black-box evasion attacks effective and efficient. In Lorenzo Cavallaro, Johannes Kinder, Sadia Afroz, Battista Biggio, Nicholas Carlini, Yuval Elovici, and Asaf Shabtai, editors, *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2019, London, UK, November 15, 2019*, pages 83–94. ACM, 2019. doi: 10.1145/3338501.3357366. URL `https://doi.org/10.1145/3338501.3357366`.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

[30] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998.

[31] Pengcheng Li, Jinfeng Yi, and Lijun Zhang. Query-efficient black-box attack by active learning. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018*, pages 1200–1205. IEEE Computer Society, 2018. doi: 10.1109/ICDM.2018.00159. URL `https://doi.org/10.1109/ICDM.2018.00159`.

[32] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=Sys6GJqxl`.

[33] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *CoRR*, abs/1605.07277, 2016. URL `http://arxiv.org/abs/1605.07277`.

[34] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*, pages 506–519. ACM, 2017. doi: 10.1145/3052973.3053009. URL https://doi.org/10.1145/3052973.3053009.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[36] Duncan Potts and Claude Sammut. Online nonlinear system identification in high dimensional environments. 11 2020.

[37] Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. Adversarial diversity and hard positive generation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA, June 26 - July 1, 2016*, pages 410–417. IEEE Computer Society, 2016. doi: 10.1109/CVPRW.2016.58. URL https://doi.org/10.1109/CVPRW.2016.58.

[38] Stuart Russell and Pter Norvig. *Artificial Intelligence: A Modern Approach (second edition)*. Prentice-Hall, 2002.

[39] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling. *Foundations and Trends in Machine Learning*, 11(1):1–96, 2018. doi: 10.1561/2200000070. URL https://doi.org/10.1561/2200000070.

[40] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Foundations and Trends in Machine Learning*, 12(1-2):1–286, 2019. doi: 10.1561/2200000068. URL https://doi.org/10.1561/2200000068.

[41] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014. URL http://dl.acm.org/citation.cfm?id=2670313.

[42] Farhana Sultana, Abu Sufian, and Paramartha Dutta. Advancements in image classification using convolutional neural network. *CoRR*, abs/1905.03288, 2019. URL http://arxiv.org/abs/1905.03288.

[43] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[44] Fnu Suya, Jianfeng Chi, David Evans, and Yuan Tian. Hybrid batch attacks: Finding black-box adversarial examples with limited queries. In *USENIX*, pages 1327–1344. USENIX Association, 2020.

[45] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. URL http://arxiv.org/abs/1312.6199.

[46] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2818–2826. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.308. URL https://doi.org/10.1109/CVPR.2016.308.

[47] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. In *ICLR*. OpenReview.net, 2018.

[48] Vladimir Vapnik. Principles of risk minimization for learning theory. In John E. Moody, Stephen Jose Hanson, and Richard Lippmann, editors, *Advances in Neural Information Processing Systems 4, [NIPS Conference, Denver, Colorado, USA, December 2-5, 1991]*, pages 831–838. Morgan Kaufmann, 1991. URL `http://papers.nips.cc/paper/506-principles-of-risk-minimization-for-learning-theory`.