Imperial College
London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Bayesian Optimization for Black-Box Evasion of Machine Learning Systems

Author:
Kenneth T. Co

Supervisor:
Luis Muñoz-González

Submitted in partial fulfillment of the requirements for the MSc degree in MSc Specialism (Machine Learning) of Imperial College London

September 2017

Abstract

Machine learning and big data algorithms have had widespread adoption in recent times, with extensive use in big industries such as advertising, e-commerce, finance, and healthcare. Despite the increased reliance on machine learning algorithms, general understanding of its vulnerabilities are still in the early stages. Because of this, there has to be a better grasp of its security implications to prevent attacks that could undermine the integrity of machine learning systems. Currently, attackers can use adversarial samples to fool even the state-of-the-art machine learning algorithms. Adversarial samples are legitimate inputs altered by adding a specially crafted perturbation, these samples retain their true class while forcing the target algorithm to misclassify them. However, current attack methods require knowledge of the target's underlying model or training data. To add to this, there is a lack of a general technique that can detect adversarial samples. I introduce a novel black-box attack against machine learning classifiers that uses Bayesian optimization. This technique is data-efficient and model-independent, two properties that existing attacks do not have. Its model-independence allows it to find adversarial regions regardless of the underlying algorithm, and its data-efficiency makes it suitable for attacking machine learning systems undetected. I demonstrate its effective against three different types of classifiers and show that it outperforms a white-box attack, fast gradient sign method (FGSM), when against a convolutional neural network (CNN). Bayesian optimization achieves an evasion rate of 99% while FGSM achieves 62% for the same perturbation setting. This implies that Bayesian optimization can be used to attack high-dimensional problems and supports the hypothesis that machine learning algorithm have low effective dimensionality. The results also show that a black-box attack through Bayesian optimization can potentially unmask adversarial samples that are not even found by the strongest white-box attackers. It has the potential to become a general framework for probing any machine learning algorithm or defense for adversarial samples.

Acknowledgements

This project would not be possible without the novels ideas, insights, guidance, banter, excitement, and encouragement from my advisor Luis.

I would also like to thank Eduardo and Leslie for welcoming me to their work environment where I was able to do research all summer long.

And finally, a huge thank you to my family for sending me to Imperial College London. Without their support, literally none of this would have been possible.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

Advances in computational and storage capabilities of computers have paved the way for the widespread use of machine learning and big data algorithms. These data-driven techniques take advantage of the plethora of information gathered from various monitoring systems and sensor networks, social media, and personal devices. These algorithms and their deployment have fundamentally transformed how basic services are being implemented today, with extensive use in big industries such as advertising, e-commerce, finance, and healthcare [4].

Despite the increased reliance and use of machine learning algorithms, general understanding of its vulnerabilities are still in the early stages. A lot of machine learning algorithms operate under the assumption of stationarity, that is, the data distributions are identically and independently distributed [4]. In an adversarial setting, attackers can violate this by generating samples with perturbations that do not follow this stationary distribution [5]. Many papers have shown crafting algorithms that generate adversarial samples that fool state-of-the-art machine learning techniques.

Adversarial samples are legitimate inputs altered by adding some perturbation. These samples retain their true class while forcing the target algorithm to misclassify them [5]. In a way, these are optical illusions for machine learning algorithms, as illustrated in **Figure 1.1**.

A machine learning classifier is a model that learns a mapping between inputs and a set of class labels. For example, a malware detector classifies files to 'benign' or 'malicious' classes. Any classifier deployed is vulnerable to probing attacks. Adversaries can use the same channels as ordinary users to gain information about the targeted system and use that

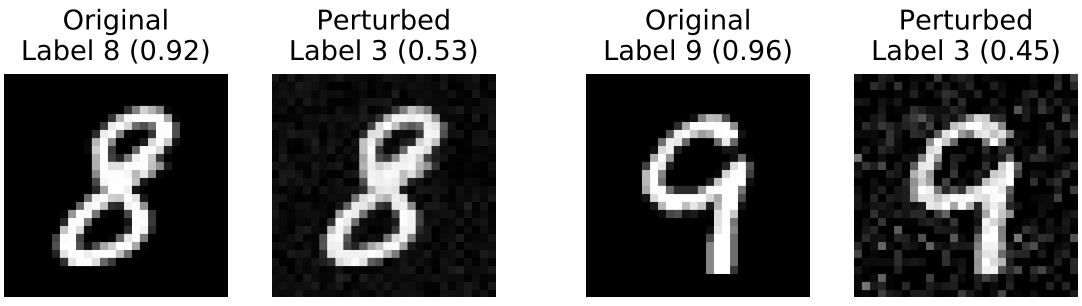

Figure 1.1: Adversarial Samples - (Left) '8' misclassified to '3'. (Right) '9' misclassified to '3'. Adversarial perturbations cause misclassification.

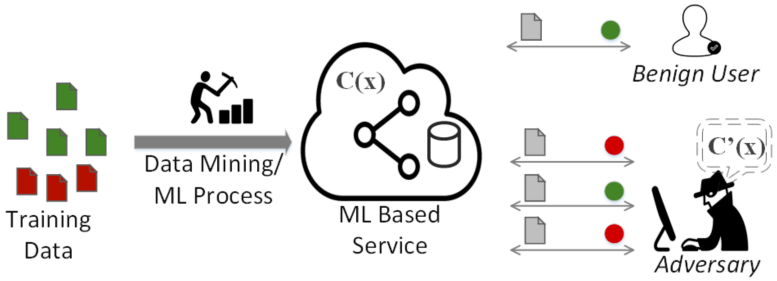

Figure 1.2: Probing Attacks - an adversary probes the black-box model C to construct a surrogate C' to later generate adversarial samples with.

knowledge to craft adversarial samples for them.
Despite the amount of research currently being done in adversarial machine learning, a lot of those methods rely on detailed knowledge of the underlying target model or access to a large enough training dataset to create an auxiliary model [5]. This limited the viability of those attacks to strong adversaries with insider knowledge. Existing black-box attacks such as Papernot et al. [6] rely on gradient-based and transferability attacks. Gradient-based attacks rely on having the gradient of the cost function of the target model to craft adversarial samples, and transferability attacks use an auxiliary model under their control to generate adversarial samples. The latter method is usually costly to train and gather data for. Additionally, no true model-independent frameworks for attack have yet been explored.

In this paper, we introduce a novel black-box attack against machine learning classifiers that uses Bayesian optimization. This technique is data-efficient and model-independent, two properties that existing attacks do not have. It has these two properties due to its use of Gaussian processes. We demonstrate that this attack can perform successful evasion against a few classifiers despite (1) having no information about the target model or what algorithm it uses and (2) having no access to any large training dataset.

The threat model we implement has a weak adversary that only knows the input and output representations of the machine learning model. We assume that the model outputs the probability vector of its prediction. The adversary's capabilities will be limited to querying the target classifier and observing the corresponding outputs.

Bayesian optimization is a sequential query-based optimization algorithm that uses statistical surrogate models, in our case Gaussian processes, to model the target classifier. We take advantage of Gaussian processes' ability to model any function to make Bayesian optimization a model-independent approach. This attack framework will be effective regardless of the target's underlying machine learning algorithm.

Defensively, this can be used as a general framework for machine learning developers to probe their models for adversarial samples. It can also be used to test the vulnerability of new or existing machine learning algorithms and defenses. This is important as current crafting algorithms and defenses are too specific in their design to find all possible adversarial samples.

In our experiments we attack three types of classifiers: convolutional neural network (CNN), logistic regression (LR), and random forest (RF) ensemble using four types of attacks: Bayesian optimization, Bayesian optimization with dimensionality reduction, fast gradient sign method (FGSM), and random attack.

The contributions of this paper are the following.

- We introduce a novel black-box attack against machine learning classifiers that uses Bayesian optimization. This algorithm is data-efficient and model-independent. We show that it is performs well against all the classifiers and is even more effective than a white-box attack against the state-of-the-art convolutional neural network.
- Because it outperforms a white-box attack, a much stronger adversary, Bayesian optimization can potentially unmask adversarial samples that are not even discoverable by current techniques.
- We perform the Bayesian optimization attack with dimensionality reduction via principal component analysis. Its failure implies that a pure Bayesian optimization attack is able to exploit the less relevant features to evade the classifier.
- The ease at which the random forest ensemble was evaded brings into question the security of the algorithm.
- The success of Bayesian optimization reinforces the hypothesis that machine learning algorithms have low effective dimensionality and shows the potential for it to be a general black-box attack regardless if inputs have a high number of features.

This paper is organized as follows.

- **Section 2** presents the background and related work for the security of machine learning, evasion attacks, and Bayesian optimization.
- **Section 3** presents the adversary threat model, formalization necessary for it, the design and parameter choices for Bayesian optimization, and the crafting algorithms we use.
- **Section 4** presents the experimental results and discusses the implications of it.
- Lastly, **Section 5** discusses the conclusions, suggest some defenses, and presents various extensions for future work.

Chapter 2

Background & Related Work

In this chapter, we give an introduction to adversarial machine learning, discuss the current literature on evasion attacks, and give motivation for why Bayesian optimization is a suitable approach for black-box attacks against machine learning systems.

- **Chapter 2.1** illustrates the broad categories of attacker motivations and methods in adversarial machine learning.
- **Chapter 2.2** goes over the current techniques being used for evasion attacks, moving towards the more realistic scenario of black-box attacks.
- **Chapter 2.3** highlights the lack of a true black-box attack and proposes Bayesian optimization as a solution, it gives more formal introduction to it as well.
- **Chapter 2.4** goes over the theoretical motivations for why Bayesian optimization is a suitable technique for black-box attacks against machine learning systems.

2.1 Security of Machine Learning

Despite advances in machine learning (ML), the general understanding on the flaws and weaknesses inherent in the algorithms and systems that it is designed on, is still in its early stages [5]. The versatility of machine learning algorithms to solve complex problems has made it into a fundamental tool for computer systems and security. However, this versatility can also be exploited by attackers to compromise the system [7].

In recent literature, machine learning algorithms have been shown to contain numerous exploits. This makes the algorithms primary weak points in the security chain, and if exploited by attackers, could compromise the entire system or service that it is a part of. Additionally, with its applications in security such as fraud, spam, and malware detection, vulnerabilities in these ML algorithms will allow adversaries to bypass these anomaly detection mechanisms [8].

It has even been shown that the state-of-art models in machine learning such as deep neural networks (DNN) have weaknesses towards adversarial samples - seemingly benign inputs that are intentionally designed to cause the learning algorithm to make mistakes [8]. These are like optical illusions for ML algorithms [9].

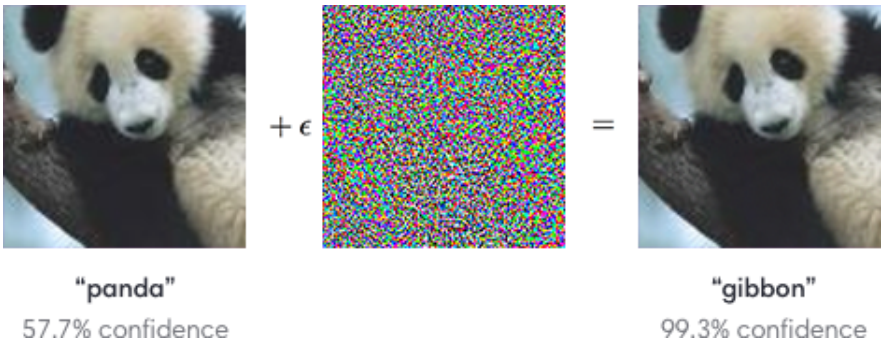

Figure 2.1: Adversarial Sample - an imperceptible adversarial perturbation overlaid on a regular image causes the classifier to misclassify the panda as a gibbon [1].

Attacks on machine learning systems such as autonomous bots, anti-virus software, and spam filters have already occurred [4]. Research in this area aim to explore the weaknesses in machine learning systems against malicious attacks. Using that knowledge, machine learning developers can better design more resilient and robust systems.

2.1.1 Adversarial Goals

Adversarial goals can be put into four broad categories according to impact on: confidentiality, integrity, availability, and privacy. The first three come from the classical CIA security model [5]. These categories can be grouped into two pairs due to their inherent similarities, (1) confidentiality and privacy, then (2) integrity and availability.

Confidentiality and Privacy
Attacks on confidentiality and privacy aim to gather information about the model or training data. The model itself can be exposed to white-box attacks, so it requires that the model and its parameters be confidential. In some contexts, it is important to maintain the privacy of training data such as with financial records, medical information, or user history.

Most machine learning techniques aim to model the underlying distribution of its training data. Hence, it learns the inherent properties of the training data. Because of this, it is difficult to guarantee complete privacy of the dataset and the individuals it represents. The data can be exposed by performing different types of queries: membership queries (whether an individual is in the dataset), recovering partially known inputs, or extracting properties of the training data using the model's outputs.

Integrity and Availability
Attacks on integrity and ability aim to control model outputs in a way that the adversary desires. By manipulating with the model's behavior and therefore its output, the integrity of the machine learning algorithm is undermined. Closely related, attacks on availability look to reduce the quality or performance of the machine learning system. While they have differences, the attack vectors are usually similar.

Integrity is a crucial factor for evaluating machine learning models as it is the basis of the performance metrics used for them. This includes metrics such as accuracy, f-scores, and false positive rates. It has been shown that the integrity of machine learning systems can be compromised by adversaries that tamper with the target model's training data or craft

specific inputs (adversarial samples) [5, 10].

Manipulating the training data could make the model learn the wrong underlying distribution and therefore perform poorly on actual data. Crafted inputs or adversarial samples are inputs where the target model misclassifies them due to errors of the target model. These have been shown to exist and occur in several machine learning models, including state-of-the-art neural networks [11, 10]. For example, a classifier could assign the wrong class to an authentic image or confidently classify a meaningless image.

Availability is about the prevention of access to a feature, making the target model inconsistent or unreliable. when some of their input features are corrupted or missing [8]. Most availability attacks corrupt the model by tampering with its training data or by reducing its confidence in ways similar to that or integrity attacks [5].

2.1.2 Types of Attacks

These attacks can be separated into two categories according to when they are deployed, either during the training phase or during the inference phase of the model.

Training Phase
Attacks during the training phase aim to influence or corrupt the model. This is done through altering the training data (poisoning attacks) or tampering with the learning algorithm (logic corruption) [5]. The former is the more common and plausible case as it requires less insider access and knowledge to the target model.

There are two general types of poisoning attacks. The first alters the training data either by inserting adversarial inputs into the existing training data. The second alters the training data directly. By doing either, the attacker can influence the behavior of the machine learning system, and degrade its overall performance. This affects its integrity and availability.

If the adversary has knowledge of the learning algorithm used by the target model, they can build a surrogate model to assist with poisoning attacks. This is done by testing their potential inputs with their surrogate model before submitting these to the target model.

Logic corruption is a more powerful attack during the training phase is when the adversary can tamper with the learning algorithm itself. Adversary that has control over the target model is clearly very powerful and extremely difficult to defend against once they have access, making it not as likely to occur [5].

Inference Phase
Attacks during the inference phase aim to produce inputs that undermine the integrity of the target model or gain more information about the target model and its training data. Unlike training phase attacks, these do not tamper with the target model [5].

Depending on the amount of information known about the target model and its training data, inference attacks can be classified as either white-box or black-box attacks.

In white-box attacks, the adversary has partial information, such as the model architecture, model parameters, or the partial or full set of training data used. This information is then

leverage to craft exploits against the target model. For example, an adversary with access to the model parameters, architecture, and training data can train a surrogate model to craft adversarial examples with.

In black-box attacks, the adversary has no information aside from that gained by querying the target model. In general, the adversary probes the model by asking about specific inputs and observing the target model's corresponding outputs [5]. Despite the scarcity of information, it has been shown that enough information can be extracted from these input-output pairs to mount black-box attacks. Black-box attacks are a more relevant threat than white-box attacks since they do not need as much knowledge or access to the targeted models.

Machine learning has many applications in computer vision and is a popular topic of research. For example, it is heavily used in self-driving cars and robotics. However, research targeting neural networks working on computer vision problems have shown the existence of adversarial samples, images that are misclassified due to perturbations that are imperceptible to humans [1, 8].

These attacks on classifiers with adversarial samples are a relevant threat because they violate the integrity of the model and are not easily detected by humans. Many papers demonstrate these "evasion attacks" against machine learning classifiers.

2.2 Evasion Attacks

Evasion attacks are a type of *inference phase* attack on classifiers with the goal of undermining the availability or integrity of the algorithm. These attacks consist of finding or generating the appropriate input that causes the target model to misclassify. For example, these can be used to force misclassification of images, as described in the previous section, or to evade detectors of malicious objects such as spam or malware [12].

Existing inference and evasion attacks have required a substantial amount of information regarding the target model or its training data. To craft adversarial examples these methods needed either detailed knowledge of the underlying model architecture and its parameters, or enough labeled training data to create a workable surrogate model [5, 6]. These limited the viability of the methods to adversaries with insider knowledge of the targeted model or to those with the capability of collecting a large number of labeled training data, both of which are niche cases.

Attacks can be separated into white-box and black-box attacks. The former assumes that the adversary has substantial internal knowledge about the target model, while the latter does not make that assumption.

2.2.1 White-Box Attacks

White-box attacks have varying degrees of access to the target model and its parameters. This strong adversary will be able to conduct very powerful attacks because of their knowledge of the target model. Realistically, it is often difficult to obtain this information and

typically requires insider knowledge or reverse-engineering. However, this scenario is still possible [5].

Majority of white-box attacks involve using a version of the fast gradient sign method suggested by Goodfellow et al. [1]. This fast gradient sign involves using the gradient of of the target f As a result of this, defenses have been made to tamper with adversarial crafting heuristics such as by masking gradients, but these techniques do not mitigate the underlying incorrect model predictions. Transferability-based black-box attacks have been shown to evade this defense [5].

Reliance on gradient-based methods or transferability attacks from a surrogate model require a strong adversary with enough information about the target model or its training data. We look to a more realistic threat of an adversary with no detailed information about the target model or its training data.

2.2.2 Black-Box Attacks

Black-box attacks are a realistic threat. Existing machine learning algorithms are vulnerable to these attacks regardless of their underlying structure. Transferability attacks between pairs of machine learning model classes are possible. However these attack require access to a large enough dataset that is representative of the target model's training data [11].

More realistic black-box attacks have been demonstrated against deep neural network classifiers. In their black-box attack, Papernot et al. [6] assumed that the adversary (i) had no information on the model architecture and parameters, and (ii) did not have access to any large training dataset. Their adversary was only capable of feeding its own queries to the target deep neural network and observing the corresponding output labels assigned.

Papernot et al. [6] is so far the only comprehensive paper published that demonstrates an effective attack against a black-box machine learning algorithm with no access to a large dataset [5], our paper adopts a similar framework to theirs. Papernot et al. [6] illustrated that their black-box attack demonstrates three key properties that made it applicable to many systems whose decision-making is based on machine learning: "(i) the capabilities required are limited to observing output, (ii) the number of labels queried is limited, and (iii) the approach applies and scales to different ML classifier types, in addition to state-of-the-art DNNs". The attack we propose shall demonstrate these three key properties as well.

As in Papernot et al. [6], the adversary has to carefully select the next query for the target classifier to gain the most information and come closer to the goal of forcing misclassification on their specific input. This search for adversarial samples can be reformulated to finding a solution for an optimization problem against the targeted machine learning model [9]. This is generally done by estimating the target model's classifier and minimizing its confidence in predicting a sample's true class label.

In an adversarial setting, the number of queries to the target model must be constrained to avoid detection by the defender, so queries to the target model (i.e. the number of function evaluations) are considered to be expensive. The most successful methods for expensive black-box functions are based on surrogate models of the objective function built incremen-

tally from the queries [13]. Hence, it is necessary use a surrogate model the target classifier's behavior for our optimization problem.

In Papernot et al. [6], their surrogate model for the targeted deep neural network was a deep neural network of their own. However, it could be argued that their's is not a true black-box attack because they knew the type of machine learning model they were attacking. They may not have the same success if the targeted model was not similar to a deep neural network. The type of machine learning algorithm being attacked can not always be known, there is therefore a need for a model-independent approach when performing black-box attacks.

2.3 Bayesian Optimization

Bayesian optimization a sequential model-based optimization algorithms. These types of models use previous observations of their target function to determine the next best point to sample or query from.

Bayesian optimization is primarily used in optimizing expensive black-box functions. It has proven to be an effective solution to various design problems and has made great impact in a wide range of research such as in sensor networks, hyperparameter tuning, reinforcement learning, robotics, and combinatorial optimization [3]. In the adversarial machine learning setting, this method could provide more insight into the target model's learned classifier function.

This method has been used to find the global optima of unknown objective functions within a design space of interest. Typically, the design space is a compact subset of its domain, but the framework is adaptable and can be applied to various kinds of domains, discrete or continuous [3]. This allows it to be used for the different kinds of feature input spaces used by machine learning models.

2.3.1 Introduction

Bayesian optimization has a prior belief over the possible objective functions and then sequentially refines this model by making queries and then updating the posterior. The posterior represents the updated belief given what was observed. Bayesian optimization consists of two primary components, the first being a probabilistic surrogate model, such as a Gaussian process, and the second being an objective function that describes its running performance [3].

The first component, a probabilistic surrogate model, consists of a prior distribution that captures our known beliefs of the unknown function and an observation model that describes the querying mechanism. A Gaussian process is the generalization of multivariate Gaussian distributions to functions and is typically used as the surrogate model for Bayesian optimization [3].

For this research we will use Gaussian processes as our surrogate model because it induces a posterior distribution over the objective function that is *analytically tractable*. This allows us to update our beliefs of what the objective function looks like after each iteration. We go

into detail on Gaussian processes in the next subsection.

The second component is a objective function that "describes how optimal a sequence of queries is". Ideally, the expected objective function value is then minimized to select an optimal sequence of queries [3]. This is usually done through an acquisition function.

The acquisition function guides the search for an optimum. Intuitively, it evaluates the utility of candidate points for the next evaluation. Acquisition functions are normally defined so that high acquisition corresponds to potentially high values of the objective function. Maximizing the acquisition function is used to select the next point at which to evaluate the function [14].

For each iteration n, Bayesian optimization uses its acquisition function to select an input $\boldsymbol{x}_{n+1}$ to query the target classifier f and then observes the output $f(\boldsymbol{x}_{n+1}) = y_{n+1}$. The prior of the statistical model is then updated to produce a posterior distribution of functions that is more representative of the observed data.

The algorithm stops once the best objective function value ceases to improve or when the algorithm reaches a maximum number of iterations (queries). The algorithm then makes a final recommendation that represents its best estimate of the optimum [3]. See **Algorithm 1** for an illustration and pseudocode of the algorithm.

Algorithm 1: Bayesian Optimization

for $n = 1, 2, \ldots,$ **do**
  select new $\boldsymbol{x}_{n+1}$ by optimizing acquisition function α
  $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x}; \mathcal{D}_n)$
  query objective function with input $\boldsymbol{x}_{n+1}$ to obtain its output y_{n+1}
  augment dataset $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\boldsymbol{x}_{n+1}, y_{n+1})\}$
  update statistical model

The Bayesian optimization framework is also very data efficient [3]. It is useful in situations where function evaluations (queries) are costly, when one does not have access to derivatives with respect to the input, or when the target function is non-convex and multimodal [3]. Bayesian optimization makes use of all the information provided by previous queries to make its search efficient.

In the next two subsections we explain the details behind Gaussian processes and acquisition functions, two key components of our Bayesian optimization algorithm.

2.3.2 Gaussian Processes

A Gaussian process (GP) is a generalization of Gaussian distributions to a distribution over functions [15]. Like how a Gaussian distribution is defined by a mean and variance, a GP is defined by a mean and covariance (kernel) function. Formally, the Gaussian process $\mathcal{GP}(\mu_0, k)$ is a nonparametric model that is fully described by a prior mean $\mu_0 : \mathcal{X} \to \mathbb{R}$ and a positive-definite kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ [3].

Definition: A *Gaussian process* is a collection of random variables, any finite number of which form a Gaussian distribution.

Informally, you can think of a function as an infinitely long vector and the GP describes their distribution. Just like how a set o n-dimensional vector can be described by an n-dimensional Gaussian distribution.

Let f be the target objective function. To compute a posterior expectation in Bayesian optimization, we need a likelihood model for samples from f, and a prior probability model on f. Think of the GP as a function that instead of returning a scalar value $f(\boldsymbol{x})$, it returns the mean and variance of a normal distribution over the possible values of f at $\boldsymbol{x}$.

Recall for a standard regression problem, we assume a normal likelihood with noise

$$y = f(\boldsymbol{x}) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

That is, we have $y \mid f \sim \mathcal{N}(f(\boldsymbol{x}), \sigma_\varepsilon^2)$.

Let $\boldsymbol{x}_{1:n} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a set of data points and $\boldsymbol{f} = f_{1:n} = \{f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)\}$. For GP regression, we write something analogous to the standard regression problem. Assume that $\boldsymbol{f}$ is jointly Gaussian and the observations $\boldsymbol{y} = y_{1:n}$ are normally distributed given $\boldsymbol{f}$. This gives the following generative model:

$$\boldsymbol{f} \mid \boldsymbol{X} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{K}) \tag{2.1}$$

$$\boldsymbol{y} \mid \boldsymbol{f}, \sigma^2 \sim \mathcal{N}(\boldsymbol{f}, \sigma^2 \boldsymbol{I}) \tag{2.2}$$

where elements of the mean vector $\boldsymbol{m}$ and covariance matrix $\boldsymbol{K}$ are given by $m_i = \mu_0(\boldsymbol{x}_i)$ and $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ respectively [3]. Equation (2.1) represents the prior distribution.

The marginalization property of Gaussian distributions is used to make calculating GPs analytically tractable. It has become a popular statistic model because of this property.

2.3.3 Acquisition Functions

As mentioned, the acquisition function guides the search for the next best values to query. Typically, we choose the points that maximize the utility measured by the acquisition function. It is defined as a function of the posterior distribution over f that describes some form of utility.

We discuss acquisition functions with the goal of minimizing the objective function. The acquisition function has to balance the trade-off between exploration and exploitation. Exploration seeks high variance, where there is large uncertainty with the model. Exploitation seeks places with low mean, where the model is confident of low values [3]. Too little exploration can cause it to get stuck at local minima, while too much exploration does not allow you to exploit observations made.

For iteration n, let $\mathcal{D}_n$ be the collection of queries and observations so far. The GP posterior gives a predictive mean $\mu_n(\boldsymbol{x})$ and predictive marginal variance function $\sigma_n^2(\boldsymbol{x})$. Define the

function γ below.

$$\gamma(\boldsymbol{x}) = \frac{\mu_n(\boldsymbol{x}) - \tau}{\sigma_n(\boldsymbol{x})}$$

We now look at different kinds of policies for acquisition function. Given $\mathcal{D}_n$, the goal is to find $\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x}; \mathcal{D}_n)$.

Improvement-based acquisition functions go for points that are likely to improve upon the current best value $f(\boldsymbol{x}_{\text{best}}) = \tau$. An example is the **probability of improvement** acquisition function,

$$\alpha_{\text{PI}}(\boldsymbol{x}; \mathcal{D}_n) = P(f(\boldsymbol{x}) > \tau) = \boldsymbol{\Phi}(\gamma(\boldsymbol{x}))$$

where $\boldsymbol{\Phi}$ is the standard normal cumulative distribution function. A more common and widely-used acquisition function is **expected improvement**,

$$\alpha_{\text{EI}}(\boldsymbol{x}; \mathcal{D}_n) = (\mu_n(\boldsymbol{x}) - \tau)\boldsymbol{\Phi}(\gamma(\boldsymbol{x})) + \sigma_n(\boldsymbol{x})\phi(\gamma(\boldsymbol{x}))$$

where ϕ is the standard normal probability density function [15].

Another type are optimistic policy acquisition functions. The more popular one used is the Gaussian process **upper confidence bound**. It is defined as,

$$\alpha_{\text{UCB}}(\boldsymbol{x}; \mathcal{D}_n) = \mu_n(\boldsymbol{x}) - \beta_n\sigma_n(\boldsymbol{x})$$

where σ_n is a hyperparameter that adjusts for exploration [3].

These acquisition functions have varying properties and will be discussed further in **Chapter 3.3.1**, where we choose the settings for our implementation of Bayesian optimization.

2.4 Motivation

Going back to black-box attacks in adversarial machine learning, recall that at the end of **Chapter 2.2** we concluded that there is a need for a surrogate model-based attack that is not reliant on the type of the machine learning algorithm being targeted. Bayesian optimization fits this criteria as it makes use of a surrogate model in Gaussian processes. Because Gaussian processes are a generalization of function distributions, it can model any unknown function [3]. This last property allows general Bayesian optimization to be effective regardless of the underlying machine learning algorithm.

The three key properties Papernot et al. [6] outline that general black-box attacks on machine learning systems should satisfy are: "(i) the capabilities required are limited to observing the output, (ii) the number of labels queried is limited, and (iii) the approach applies to different ML classifier types, in addition to state-of-the-art DNNs". Bayesian optimization can be set up to have property (i). It satisfies the remaining properties (ii) and (iii) respectively because it is data efficient and it does not need to have knowledge of the targeted ML classifier. This makes Bayesian optimization a natural fit for black-box adversarial machine learning.

2.4.1 Low Effective Dimensionality

Bayesian optimization finds the best sequence of queries to the target model to gather information. For high dimensions, this problem quickly becomes intractable as the search space increases exponentially with the number of features and possible values. To illustrate, consider a target model with M input features each having K possible values. The number of possible inputs to query in the search space is therefore M^K. The intractability is even more apparent for input features in the continuous domain. This is why standard black-box numerical optimization techniques such as quasi-Newton methods are not appropriate for this problem due to high dimensionality. Surrogate models, like Bayesian optimization, are the better choice for dealing with black-box functions having high-dimension inputs [13].

Bayesian optimization has had consistent success with problems of moderate dimensionality; with state-of-the-art performance being achieved from tuning up to 76 parameters in combinatorial optimization problems. For high dimensionality, most features do not actually change the objective function in a significant way for certain classes of problems. Some papers have shown this including "hyperparameter optimization for neural networks and deep belief networks and automatic configuration of state-of-the-art algorithms for solving NP-hard problems" [3]. In effect, these problems with high dimensionality have *low effective dimensionality*.

To take advantage of low effective dimensionality, there has been research on the use random feature selection by Bergstra and Bengio [16] and random embeddings by Wang et al. [17]. Aside from these, Goodfellow et al. [1] demonstrated that adversarial samples are caused by machine learning models being too linear in high-dimensional space by applying linear perturbations to create adversarial samples. They state that linearity makes the models easy to train, but susceptible to adversarial samples. This linearity suggests that machine learning algorithms learn classifiers that have low effective dimensionality. If this hypothesis holds, Bayesian optimization will be an effective technique for generating adversarial samples against machine learning algorithms even if their inputs live in high-dimension space.

The phenomena of the classifier being too linear is illustrated in **Figure 2.2**. If class A represents a benign sample and class B a malicious one, then the adversary can have malicious samples identified as benign in the leftmost adversarial region. Goodfellow et al. [1] have shown that this occurs in high-dimension spaces.

We hypothesize that machine learning algorithms have low effective dimensionality. As a result, this makes them vulnerable to Bayesian optimization modeling their learned classifier functions and generating adversarial samples through that.

2.4.2 Model Independence

At the moment, [6] is the only effective black-box attack we can compare to, but it requires prior knowledge of the target's machine learning algorithm. In their case, it was a deep neural network (DNN). If the target model is a DNN or a similar classifier that is vulnerable to the same adversarial samples, then their may be a more efficient attack overall. However, decision trees (DT) and k-nearest neighbors (KNN) are fairly robust against adversarial samples from DNNs [3]. The researchers noted that DTs and KNNs are non-differentiable

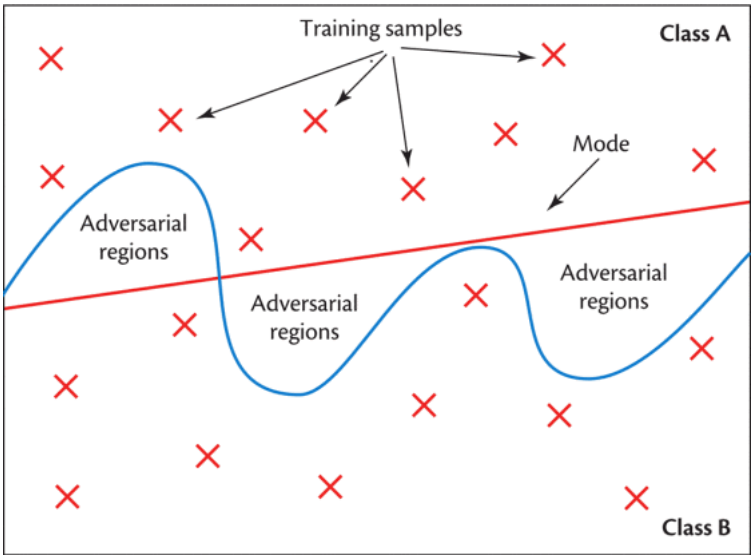

Figure 2.2: Adversarial Regions - in a simplified representation, the plane represents the input feature vector space. Crosses are samples. Classes A and B are separated by the curve which is the *true* decision boundary. The red line represent a classifier's *learned* decision boundary [2].

machine learning algorithms whereas DNNs are not. Differences in differentiability, the use of gradients, or convexity of the learned classifier function among other things greatly affect the effectiveness of the method suggested by Papernot et al. [6]. Additionally, new classes of machine learning algorithms may be discovered that behave completely differently from existing ones.

The limited uses of current black-box attacks and the plethora of research with white-box attacks [5] suggests that a different approach needs to be taken. Most adversarial sample crafting algorithms are based of the fast-gradient sign method suggested by Goodfellow et al. [1]. This may not always be effective or transferable to all other machine learning algorithms.

The robustness of Bayesian optimization, and Gaussian processes in particular, allows it to be effective against any target model regardless of the underlying machine learning algorithm. This is why Bayesian optimization is a promising technique for performing true black-box attacks. If successful, this method can become a general-purpose black-box attack that will be used to probe machine learning models for adversarial samples. This also has a potential application as a security audit for machine learning algorithms.

Chapter 3

Design

In this chapter, we lay out the framework for attacking the target model from the adversary's perspective, talk about Bayesian optimization in an adversarial setting, and finally go over several adversarial crafting algorithms.

- **Chapter 3.1** defines the theoretical framework needed to quantify the adversary's goals.
- **Chapter 3.2** outlines the threat model used and lists the assumptions made. This is the primary section for understanding the problem setting and the adversary's knowledge, capabilities, and goals.
- **Chapter 3.3** discusses Bayesian optimization in an adversarial setting, chooses the hyperparameters, and lists its strengths and limitations.
- **Chapter 3.4** details the various algorithms that we propose to craft adversarial samples. This sets up the experiments we will run in the following chapter.

3.1 Theoretical Framework

To properly measure the adversary's goals, it is essential to formally define the problem. This paper adopts the theoretical framework laid out in Wang et al. [18] with some minor changes. In this section, we work towards formal definitions for learned classifiers, true class labels, and adversarial samples.

Given a classification problem with K classes, let $\boldsymbol{x}$ be the input sample space of the feature vectors and Y be the output space representing a categorical set. For simplicity, let the class labels be integers $1, 2, \ldots, K$.

Definition 1. A *learned classifier* f is a function from $f : X \to Y$. For all $\boldsymbol{x} \in X$ and $1 \leq k \leq K$, let $f_k(\boldsymbol{x})$ denote the probability that the classifier believes $\boldsymbol{x}$ belongs to the k-th class.

From the definition, $[f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})] \in [0, 1]^K$ is the output probability vector for $\boldsymbol{x}$ that corresponds to the learned classifier f. Note that the classifier-assigned label is also given by $f(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x})$. This is the classifier learned by the machine learning algorithm.

Now the *true class* of an object is typically determined by a human observer. Wang et al. describes an *oracle* with the following definition.

Definition 2. An *oracle* represents a decision process generating ground truth labels for a task of interest. Each oracle is task-specific, with finite knowledge and noise-free.

Therefore, the goal of machine learning is to train a learning-based classifier $f : X \to Y$ to approximate an oracle classifier $\tau : X \to Y$ for the same classification task.

Definition 3. Given a classification task and its oracle $\tau : X \to Y$, the *true class label* of $\boldsymbol{x} \in X$ is $\tau(\boldsymbol{x})$.

Definition 4. Given an oracle $\tau : X \to Y$ and a learned classifier $f : X \to Y$ for the same task, an *adversarial sample* $\boldsymbol{x} \in X$ satisfies $f(\boldsymbol{x}) \neq \tau(\boldsymbol{x})$.

For most classification tasks like image recognition, the oracle τ are humans who observe and label the objects. In most literature, the idea for generating adversarial samples is to begin with a correctly classified sample and then apply to it a specifically crafted perturbation to force misclassification. Given a target classifier f, the adversary begins with $\boldsymbol{x} \in X$ where $f(\boldsymbol{x}) = \tau(\boldsymbol{x})$, then their goal is to produce an $x' \in X$ such that $f(\boldsymbol{x'}) \neq \tau(\boldsymbol{x})$ and $d(x, x') < \epsilon$ for some small $\epsilon > 0$ [18].

$d(\cdot, \cdot)$ is a function measuring the distance between two points in $\boldsymbol{x}$. The condition $d(x, x') < \epsilon$ means that $\boldsymbol{x}$ and x' are close enough, which would imply that $\tau(\boldsymbol{x}) = \tau(\boldsymbol{x'})$. The true class label is unchanged with the adversarial sample generated. Combining the two conditions $f(\boldsymbol{x'}) \neq \tau(\boldsymbol{x})$ and $\tau(\boldsymbol{x}) = \tau(\boldsymbol{x'})$ implies that $f(\boldsymbol{x'}) \neq \tau(\boldsymbol{x'})$. Therefore making x' an adversarial sample.

Finding $x' \in X$ subject that satisfies the two conditions above can be reformulated to finding a small enough perturbation $\boldsymbol{r} \in X$ such that $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{r}$ satisfies the conditions. You will still have $f(\boldsymbol{x} + \boldsymbol{r}) = f(\boldsymbol{x'}) \neq \tau(\boldsymbol{x})$, but the distance constraint $d(x, x') < \epsilon$ can be replaced with the condition $\|\boldsymbol{r}\| \leq \delta$ for some small $\delta > 0$.

Definition 5. $\boldsymbol{r} \in X$ is an *adversarial perturbation* if $(\boldsymbol{x} + \boldsymbol{r})$ is an adversarial sample.

It then becomes clear that the adversary's goal would be to find adversarial perturbations.

3.2 Threat Model

In this section, we outline the threat model using the framework suggested in Papernot et al. [5] and implemented in [6]. To explore the adversary's mindset in performing an evasion attack, it is necessary to make clear their knowledge, goals, and capabilities with respect to the targeted model.

The adversary wants to generate adversarial samples, by adding small perturbations to legitimate inputs, to fool the target classifier. To illustrate a black-box attack on an unknown target classifier, we must assume that the adversary can only query the target classifier and observe the corresponding output probability vector. The adversary has no knowledge of the

internal details of the target classifier, this includes the type of algorithm it uses, its training data, design, and settings.

3.2.1 Adversary's Knowledge

- The adversary has knowledge of the target classifier's input representation, expected outputs, and the number of class labels.
- The adversary does not know anything about the inner workings of the target classifier (e.g. learning algorithm, training data, etc.).
- The targeted machine learning model is a black-box image classifier that has finished training and outputs class probability vectors.

The primary reason for image recognition is that it is one of the most studied classification tasks in adversarial machine learning, and it is used in most research papers [5]. Identifying true class labels for images is also effortless for humans, making adversarial samples easy to identify in-person.

Because this project is a proof of concept, the simple image classification task is sufficient. Images come in easy-to-process arrays of pixel intensities and there are benchmark datasets such as MNIST [19]. Other classification tasks like malware and spam detection are more complicated because their features vary and the classification of malware and spam is not as straightforward, needing more time and domain knowledge to properly implement.

3.2.2 Adversary's Capabilities

- The adversary can query any input to the target classifier and observe the corresponding output probability vector.
- The adversary has access to some legitimate inputs that are correctly classified by the target model.
- The adversary has access to an oracle τ since he can look at an image and identify its true class label.

In formal terms, let $f : X \to Y$ be the learned classifier of the target model with K class labels. The adversary can query the model any $\boldsymbol{x} \in X$ and observe the corresponding output probability vector $[f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})]$, which also implies the class label $f(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x})$.

Practical applications of image classifiers need to output the probability vector to show the classifier's confidence in its predictions, so it is not an unrealistic scenario. This choice of output might be contentious because it leaks information, but without this, the attack with Bayesian optimization is near impossible as nothing else is known about the target classifier.

Masking the output probability vector is a possible defense that is discussed later in **Chapter 3.3.2** for the limitations of Bayesian optimization.

3.2.3 Adversary's Goals

- The adversary wants to find inputs that the target model classifies to any class different from the correct one.

In formal terms, the adversary wants to find adversarial samples. That is, $\boldsymbol{x} \in X$ such that $f(\boldsymbol{x}) \neq \tau(\boldsymbol{x})$ where τ is the oracle. There may be legitimate inputs that the target model misclassifies. However, there is no consistent method for finding or generating these given what the adversary knows about the target model.

The primary way to find adversarial samples, as discussed in sections 3.1, is to generate them from correctly classified inputs. We've shown that it is sufficient to find a small adversarial perturbation $\|\boldsymbol{r}\| < \delta$ such that $f(\boldsymbol{x} + \boldsymbol{r}) \neq \tau(\boldsymbol{x})$, for an $\boldsymbol{x}$ satisfying $f(\boldsymbol{x}) = \tau(\boldsymbol{x})$.

If the original predicted class is $f(\boldsymbol{x}) = l$, then the adversarial sample we want, $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{r}$, must satisfy $f(\boldsymbol{x'}) \neq l$. This implies that $\max_{k \neq l} f_k(\boldsymbol{x'}) - f_l(\boldsymbol{x'}) > 0$ [4].

Given a correctly classified $\boldsymbol{x}$, define the function $\Omega_{\boldsymbol{x}}(a) = \max_{k \neq l} f_k(\boldsymbol{x} + \boldsymbol{a}) - f_l(\boldsymbol{x} + \boldsymbol{a})$. Hence, if we write $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{r}$, then $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) = \max_{k \neq l} f_k(\boldsymbol{x'}) - f_l(\boldsymbol{x'})$.

So with an objective function $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) = \max_{k \neq l} f_k(\boldsymbol{x'}) - f_l(\boldsymbol{x'})$, the adversary's goal is to find $\boldsymbol{r}$ so that $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) > 0$. A stronger condition is to solve the following optimization problem.

$$\begin{aligned} \max_{r} \quad & \Omega_{\boldsymbol{x}}(\boldsymbol{r}) \\ \text{s.t.} \quad & \|\boldsymbol{r}\| < \delta \end{aligned} \tag{3.1}$$

This will be the goal of the Bayesian optimization. Larger $\Omega_{\boldsymbol{x}}(\boldsymbol{r})$ leads to misclassification with higher confidence, but $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) > 0$ is sufficient to force misclassification. This is important to note because it may be unlikely to achieve global optima with Bayesian optimization when working in very high-dimension space. For this project, it is enough to have $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) > 0$.

Indiscriminate & Targeted Evasion
In this project, the adversary does an *indiscriminate evasion* since the goal is to force misclassification to *any other* class different from the correct one. If the adversary wanted to force misclassification to a specific class $m \neq l$, a *targeted evasion*, then the objective function would instead be $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) = f_m(\boldsymbol{x'}) - f_l(\boldsymbol{x'})$ [4].

At the moment, the adversary does not have a heuristic for which class to force misclassification towards, so targeted evasion is set aside for future work.

Secondary Goals

- The adversary wants to minimize the number of queries made to the target classifier to avoid detection.
- The adversary wants the adversarial samples to be convincing to human observers to avoid raising suspicion.

These secondary goals are outlined because they are desirable but not easy to measure. The primary goal of the adversary is to generate adversarial samples, but it is also important to

avoid detection and raising suspicion. If the owner of the targeted model becomes aware of any attacks on their system, they will deploy countermeasures such as blocking the adversary's access.

At the moment, there are no clear measures for how intrusive the adversary can be. The secondary goals correspond to the two ways the adversary could get caught: (1) from the unusual amount and frequency of queries sent or (2) from the obvious manipulation of the input queries.

For the first risk, the adversary can space out his queries and mix in some non-malicious ones. In the analysis portion of our results, chapter 4, we address the number of queries and compare our results to [6].

For the second risk, it depends on whether or not the adversarial perturbations generated are noticeable to humans. The perturbation could be unnoticeable, noticeable but unobstructive, or clearly obstructive. Large perturbations that are clearly obstructive of the image make the true class label ambiguous, this is not desirable for the adversary.

To measure this rigorously, we would need to survey human subjects to classify the images they see and to gauge the amount of noise they perceive. An example of this was done in Papernot et al. [8], but we currently do not have the time nor resources to conduct that survey at the moment.

To achieve the latter secondary goal, we restrict the magnitude of alterations (perturbations) made on the original input sample. This is done by using a metric (e.g. L_1 or L_2 norm for inputs in $\mathbb{R}^n$) to minimize the perturbation $\|\boldsymbol{r}\| < \delta$. For the optimization problem, this will manifest as an additional box constraint on the perturbation $\boldsymbol{r}$.

With the design and threat model set, we can move on to the implementation and design of the experiment.

3.3 Bayesian Optimization

In this section, we update the Bayesian optimization algorithm with the adversary's goals in mind. We then list our design choices for the algorithm's hyperparameters and other settings. We then list the strengths and limitations of Bayesian optimization in the general context and how it affects its use in adversarial machine learning. These set up our discussion in the following section for how the adversary can generate adversarial samples using various crafting algorithms.

We will use Gaussian processes as our surrogate model as discussed in the background **Chapter 2.3**, because it induces a posterior distribution over the objective function that is *analytically tractable*.

The objective function that we are trying to maximize is $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) = \max_{k \neq l} f_k(\boldsymbol{x}') - f_l(\boldsymbol{x}')$ from equation (3.1), with the box constraint $\|\boldsymbol{r}\| < \delta$.

$\boldsymbol{x}$ is the correctly classified sample, with predicted class label $f(\boldsymbol{x}) = l$, that the adversary wants to evade the classifier with, $\boldsymbol{r}$ is the perturbation generated, and $\delta > 0$ is an upper bound for the perturbation to ensure that the true class labels of $\boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{r}$ are still the same.

The goal is to have $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) > 0$. Since $\boldsymbol{x}$ is already given and fixed, we search in terms of $\boldsymbol{r}$. Though we query the target classifier with $\boldsymbol{x} + \boldsymbol{r}$.

Given $\boldsymbol{x}$ and an initial perturbation r_1, we have an updated Bayesian optimization algorithm.

Algorithm 2: Bayesian Optimization (Adversarial Setting)

1: **for** $n = 1, 2, \ldots, M$ **do**
2: select new $\boldsymbol{r}_{n+1}$ by optimizing acquisition function α
$\boldsymbol{r}_{n+1} = \arg\max_{\boldsymbol{r}} \alpha(\boldsymbol{r}; \mathcal{D}_n)$
3: query target classifier by getting output probability vector of $\boldsymbol{x} + \boldsymbol{r}_{n+1}$
this gives us the value of $\Omega_{\boldsymbol{x}}(\boldsymbol{r}_{n+1})$
4: augment dataset $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\boldsymbol{r}_{n+1}, \Omega_{\boldsymbol{x}}(\boldsymbol{r}_{n+1}))\}$
5: update $\mathcal{GP}$ prior and posterior distributions

$\mathcal{GP}$ here stands for the Gaussian process. The algorithm stops when the perturbation allows evasion of the classifier, $\Omega_{\boldsymbol{x}}(\boldsymbol{r}_{n+1}) > 0$, or when the maximum number of iterations M is reached.

The adversary only interacts with the target model when querying on step 3 for the probability vector $[f_1(\boldsymbol{x} + \boldsymbol{r}_{n+1}), f_2(\boldsymbol{x} + \boldsymbol{r}_{n+1}), \ldots, f_K(\boldsymbol{x} + \boldsymbol{r}_{n+1})]$. This allows us to compute $\Omega_{\boldsymbol{x}}(\boldsymbol{r}_{n+1}) = \max_{k \neq l} f_k(\boldsymbol{x} + \boldsymbol{r}_{n+1}) - f_l(\boldsymbol{x} + \boldsymbol{r}_{n+1})$.

3.3.1 Settings

We discuss the hyperparameter choices for the Gaussian process and acquisition function as they are integral for implementing Bayesian optimization.

Design choices involving the maximum number of iterations, number of initial points, and the distance metric used to measure adversarial perturbation are specific to the type of input data used. These particular settings are discussed in the latter portion of **Chapter 4.1** after we describe the dataset used in the experiments.

Gaussian Process
The ability of Gaussian processes (GP) to model a rich distribution of functions rests on its kernel function. The kernel function controls important properties of the function distribution such as smoothness, differentiability, periodicity, and amplitude [3].

Typically, any prior knowledge of the target function is encoded in the kernel's hyperparameters. Getting the kernel function wrong will result in models that can not accurately model the target. However, since the adversary has little to no knowledge of the target model's internal machine learning algorithms, the adversary has to adopt a more general kernel

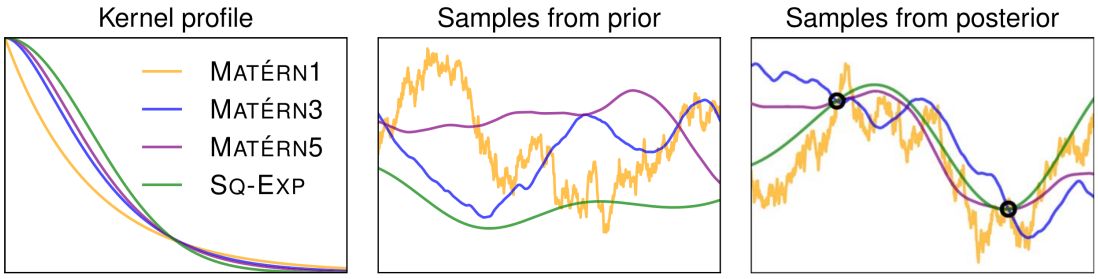

Figure 3.1: Kernel Profiles - "(Left) Visualization of various kernel profiles. The x-axis represents the distance $r > 0$. (Middle) Samples from GP priors with the corresponding kernels. (Right) Samples from GP posteriors given two data points (black circles). The sharper drop in the Matérn1/2 kernel leads to rough features in the associated samples, while samples from a GP with the Matérn3/2 and Matérn5/2 kernels are increasingly smooth" [3].

function [20].

The following are the most commonly used kernels, labeled by the smoothness parameter and omitting a factor of 1/2,

$$k_{\text{MATÉRN1/2}}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0^2 \exp(-r) \tag{3.2}$$

$$k_{\text{MATÉRN3/2}}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0^2 \exp(-\sqrt{3}r)(1 + \sqrt{3}r) \tag{3.3}$$

$$k_{\text{MATÉRN5/2}}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0^2 \exp(-\sqrt{5}r)\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \tag{3.4}$$

$$k_{\text{sq-exp}}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0^2 \exp\left(-\frac{1}{2r^2}\right) \tag{3.5}$$

where $r^2 = (\boldsymbol{x} - \boldsymbol{x'})\boldsymbol{\Lambda}(\boldsymbol{x} - \boldsymbol{x'})$ and $\boldsymbol{\Lambda}$ is a diagonal matrix of d squared length scales θ_i^2. The profiles of these kernels are illustrated in **Figure 3.1**.

In our research we will be using the GPyOpt library to implement the constrained Bayesian optimization. Their authors have set it up so that the GP takes a more model-independent approach with regards to the kernel. They implement their proposed kernel function, the automatic relevance determination (ARD) Matérn 5/2 kernel citing that other common choices like the ARD squared exponential kernel as "unrealistically smooth for practical optimization problems" [20].

$$k_{\text{MATÉRN5/2}}(\boldsymbol{x}, \boldsymbol{x'}) = \theta_0^2 \exp(-\sqrt{5}r)\left(1 + \sqrt{5}r + \frac{5}{3}r^2\right)$$

The Matérn 5/2 kernel results in twice-differentiable functions, an assumption that corresponds to those made by popular black-box optimization algorithms like quasi-Newton methods, without requiring the smoothness of the squared exponential.

For the remaining hyperparameters of the GP, they marginalize over hyperparameters and compute the integrated acquisition function. With this, they blend acquisition functions arising from samples from the posterior over the GP hyperparameters and have an estimate of the integrated expected improvement [20]. This is assuming that the acquisition function is

the Expected Improvement (EI), which is a choice that we justify next.

Acquisition Function
In the adversarial setting, it will not be practical to try out different acquisition functions and measure each of their performances. For the acquisition function, the two most popular choices are to either optimize the expected improvement (EI) over the current best result or the Gaussian process upper confidence bound (UCB). EI and UCB have both been shown to be effective and data-efficient in black-box optimization problems [20].

Recall from **Chapter 2.3.1**, that expected improvement is given by,

$$\alpha_{\text{EI}}(\boldsymbol{x}; \mathcal{D}_n) = (\mu_n(\boldsymbol{x}) - \tau)\boldsymbol{\Phi}(\gamma(\boldsymbol{x})) + \sigma_n(\boldsymbol{x})\phi(\gamma(\boldsymbol{x}))$$

and the Gaussian process upper confidence bound is defined as,

$$\alpha_{\text{UCB}}(\boldsymbol{x}; \mathcal{D}_n) = \mu_n(\boldsymbol{x}) - \beta_n\sigma_n(\boldsymbol{x})$$

Snoek et al. [20] have found the EI criterion to perform well in minimization problems, require no tuning of its own parameters, and to be better-behaved than UCB. Expected improvement also converges near-optimally. Other references such as [14] and [3] also suggest the use of expected improvement in the general case.

Prior Knowledge
We assume the adversary has extremely limited internal knowledge of the targeted model as we described in **Chapter 3.2.1**. However, in the current real-world environment, for certain classification tasks there is a limited number of machine learning algorithms that could achieve state-of-art performance. Hence, the adversary could infer certain properties about their targeted model and craft their black-box attacks appropriately.

For example, in image classification, the state-of-the-art in machine learning are convolutional neural networks. Knowing that their target achieves state-of-art performance, the adversary can guess the use of neural networks and deploy attacks like Papernot et al. [6] that specifically target neural networks. Or if no such method like this is available, they can incorporate their guess or belief of the target model into the kernel function or other hyperparameters of the Bayesian optimization model.

This is something that we will not do at the moment because we want to demonstrate that a truly blind Bayesian optimization attack is feasible and perhaps even effective against a targeted black-box classifier for any general classification task. Also, the type of guessing illustrated above may not be helpful in the case where the target model implements their own novel machine learning algorithm or if several families of machine learning algorithms all could achieve similar state-of-the-art performance in that classification task.

3.3.2 Strengths & Limitations

We outline the general strengths and limitations of Bayesian optimization in the context of adversarial machine learning.

Strengths

- Allows for black-box attacks when nothing is known about the internal machine learning algorithm of the target model.
- Very data efficient, requiring only a few function queries to find an optimum.

These two are great strengths in an adversarial setting where information on the targeted system is very scarce or unavailable. The first strength is very crucial because there are not many methods that can boast this capability; it allows a black-box attack to be possible in the first place. It is this model's primary advantage over Papernot et al. [6].

One could argue that adversarial samples generated by machine learning models are transferable to others. However it has also been shown that certain types of machine learning algorithms are resilient towards adversarial samples of other types [1]. Additionally, more novel machine learning algorithms could be developed in the future that behave in complete different ways (from existing ones). Thus, relying on gradient-based or transferability attacks can not always lead to a general black-box attack; cementing the need for a model-independent approach like Bayesian optimization.

Because Bayesian optimization is model-independent, it can be used defensively as a tool to audit the vulnerability of new or existing machine learning algorithms and defenses to adversarial samples. Existing defenses mask only a few vulnerabilities without addressing the underlying error of the algorithm. The statistical nature of Bayesian optimization allows it to reveal these adversarial regions so long as they exist.

The second strength helps the adversary avoid detection by minimizing the number of queries he needs to make. This second strength was discussed in the adversary's secondary goals in **Chapter 3.2.3**. Current evasion attacks involving surrogate models in adversarial machine learning need to train an entire machine learning model to estimate the target model [8]. This usually requires large amounts of training data that is representative of the one used by the target model or, as done in Papernot et al. [6], the generation of data through sampling methods. In the latter, the generated samples may not even be accurate. Either way, building such a dataset still requires querying the target model significantly many more times.

The number of iterations in a typical Bayesian optimization is less than a hundred to be efficient and effective [3]. Bayesian optimization slows down for a higher number of data points as we later explain when discussing limitations. This will be for each adversarial sample the adversary tries to create. Hence, given a small set of samples that need evasion, Bayesian optimization will be the more efficient technique with regards to the number of queries, as current methods need to train entire machine learning models that require more data to perform well.

Limitations

- Getting the function distribution (Gaussian process) of the target classifier wrong would lead to a very inaccurate model.
- Does not scale to high dimensions or high numbers of samples.
- Relies on the output being a probability vector rather than just a class label.

These first two limitations mirror the first two strengths. The flexibility of Gaussian processes could lead to the first limitation if it models the wrong function distribution.

The kernel function of the Gaussian process can model functions with properties that differ greatly from that of the target classifier. We mitigate this by having a general approach to optimization as discussed in **Chapter 3.3.1**. By using the Matérn 5/2 kernel we assume twice differentiability like in popular black-box optimization techniques such as Quasi-Newton methods, but without the need for smoothness.

By taking this conservative approach, the Gaussian process may not be able to capture and capitalize on all the properties of the target function, but it will not be inaccurate either. Recall in **Chapter 3.2.3** that the adversary's goals does not yet require evasion with high confidence. We have stated that it is enough to satisfy the weaker condition of achieving evasion first, and we hypothesize that this can be done with a more general kernel function.

The second limitation is because *for each iteration* the Bayesian optimization searches for the maximum of an acquisition function and recomputes the posterior of the Gaussian process distribution. For the acquisition function, we find the global maximum according to the expected improvement (in our case) while incorporating the new information from the last query that was just made. Although the acquisition function is cheap to evaluate, a high number of dimensions does not make the search trivial. Current methods for optimizing this include discretization, adaptive grids, and divided rectangles [3].

Updating the posterior of the Gaussian process is affected by high dimensionality and a high number of samples. This requires $O(n^3)$ where n is the training set size or the amount of queries already made. This n increases as we make more queries and gather their outputs. This makes it infeasible for the Bayesian optimization to take in a lot of data points at once. The good thing is that Bayesian optimization is relatively effective without needing that many evaluations (under 100). There are also techniques involving distributed computing and sparse Gaussian processes to deal with large training datasets if this becomes a problem later [3].

The issue of high dimensionality for Bayesian optimization is mitigated in certain classes of problems. Papers have shown that most features do not actually change the objective function in a significant way for neural networks and deep belief networks, only a few do [3]. Recall also that adversarial samples are possible because machine learning classifiers are too linear [1]. This is encouraging for the attacker as that implies the targeted models may have low effective dimensionality.

Another minor limitation related to this is that an entire implementation of Bayesian optimization is needed to generate adversarial perturbations for a single input sample $\boldsymbol{x}$. This may become inefficient if the adversary needs to evade using a very large set of input samples. Though most of the time, from the defender's perspective, only a few malicious samples need to get through to compromise the system.

The third limitation is raised because Papernot et al. [6] has shown that is possible to perform black-box attacks with only the output class labels. Bayesian optimization needs the quantitative continuous output to be able to reasonably perform well. In general, the output class label can always be expected, but not the output probability vector. However we ar-

gue that applications of classification algorithms in decision-making systems typically would want a measure of confidence from the classifier, so it is not unreasonable to expect the probability vector to be a part of the output.

This vulnerability for machine learning systems can be remedied by (1) showing only the output class label and its probability value, (2) masking the probabilities with more qualitative measure such as "high confidence" or "low confidence", or (3) simply not showing the probabilities at all. We revisit this defense in **Chapter 5**.

3.4 Crafting Algorithms

In this section, we list the different crafting algorithms available to generate adversarial samples. These algorithms are what we evaluate in our experiments. To begin, we summarize a general black-box attack algorithm from adversary's perspective.

Let $f : X \to Y$ be the learned classifier of the target model with K class labels and $\boldsymbol{x}$ be a correctly classified input. That is, $f(\boldsymbol{x}) = l$ where l is the true class label of $\boldsymbol{x}$.

Write $[f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})] \in [0,1]^K$ as the output probability vector for $\boldsymbol{x}$ and the classifier-assigned label is given by $f(\boldsymbol{x}) = \arg\max_k f_k(\boldsymbol{x})$.

The adversary wants to find a small adversarial perturbation $\boldsymbol{r}$ such that the adversarial sample generated from that $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{r}$ evades the classifier. That is, $f(\boldsymbol{x'}) \neq l$, or equivalently, $f_l(\boldsymbol{x'}) < f_k(\boldsymbol{x'})$ for some $k \neq l$.

This evasion problem and its boundary conditions are summarized by the following optimization problem.

$$\begin{aligned} \min_{r} \quad & \Omega_{\boldsymbol{x}}(\boldsymbol{r}) \\ \text{s.t.} \quad & \|\boldsymbol{r}\| < \delta \end{aligned}$$

Where $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) = \max_{k \neq l} f_k(\boldsymbol{x'}) - f_l(\boldsymbol{x'})$ and it is sufficient for the adversary to achieve evasion, or $\Omega_{\boldsymbol{x}}(\boldsymbol{r}) > 0$, rather than the stronger condition of finding a global maximum of the function.

On each iteration, the adversary queries the target classifier with its perturbed input $\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{r}$ to get the output probability vector $[f_1(\boldsymbol{x'}), f_2(\boldsymbol{x'}), \ldots, f_K(\boldsymbol{x'})]$ and classifier-assigned label $f(\boldsymbol{x'})$. So given an input $\boldsymbol{x}$ that is correctly classified by the target model, the adversary does the following.

Algorithm 3: General Query-Based Black-Box Attack

1: **for** $n = 1, 2, \ldots, M$ **do**
2: generate $\boldsymbol{r}_{n+1}$ using crafting algorithm ▷ where $\boldsymbol{r}_{n+1}$ satisfies $\|\boldsymbol{r}\| < \delta$
3: query target classifier with $\boldsymbol{x} + \boldsymbol{r}_{n+1}$
4: get resulting output and update model

Stopping when evasion is achieved in step 3 or when the maximum number of iterations M is reached.

This section explores the following crafting algorithms that the adversary can use for their step 2.

1. **Random:** Perturbations are generated randomly.
2. **Bayesian Optimization (BO):** Perturbations are generated only with Bayesian optimization.
3. **BO with Dimensionality Reduction:** Dimensionality reduction is applied first, then perturbations are generated with Bayesian optimization.
4. **Fast Gradient Sign Method:** White-box attack that makes use of the target model's gradients to generate adversarial perturbations.

Random is a reasonable lower baseline for black-box attacks; anything worse is not an effective method. Fast gradient sign method (FGSM) is a white-box attack that a stronger adversary will be able to perform. Matching the results of a white-box attack with a black-box attack will exceed expectations, and surpassing it will be even better. Realistically, we expect the second algorithm (Bayesian optimization) to do be better than random, but worse than FGSM.

To make up for Bayesian optimization's limitations when dealing with inputs from high-dimension spaces, we experiment with dimensionality reduction. This assumes a stronger adversary that has access to a sizable amount of labeled data that is representative of the target classifier's training data. It is worth exploring to see if Bayesian optimization attacks can be enhanced by other techniques.

3.4.1 Random

This is a simple algorithm that randomly generates the perturbation $\boldsymbol{r}$ such that it is within the boundary conditions $\|\boldsymbol{r}\| < \delta$. Depending on the metric or norm used, the boundary conditions can be satisfied by dividing the generated vector by a scalar.

Since there is no information on the underlying machine learning algorithm of the target model, it makes sense to generate $\boldsymbol{r}$ by sampling from a random multivariate uniform distribution. There are other methods to do random sampling, but this is the simplest one that makes the least number of assumptions. This algorithm is illustrated below.

Algorithm 4: Random

1: **for** $n = 1, 2, \ldots, M$ **do**
2: randomly generate $\boldsymbol{r}_{n+1}$ ▷ where $\boldsymbol{r}_{n+1}$ satisfies $\|\boldsymbol{r}\| < \delta$
3: query target classifier with $\boldsymbol{x} + \boldsymbol{r}_{n+1}$
4: **if** $f(\boldsymbol{x} + \boldsymbol{r}_{n+1}) \neq l$ **then return** $\boldsymbol{x} + \boldsymbol{r}_{n+1}$
5: get resulting output and update model

3.4.2 Bayesian Optimization (BO)

In the first few iterations, Bayesian optimization has little to no data points to take advantage of. To set it up better, we need a few evaluations at some initial points for $\boldsymbol{r}$. These are typically chosen randomly, since nothing yet is known about the target classifier. The number of initial points N depends on the input data and its dimensionality. Choosing N is discussed in **Chapter 4.1.2**.

So the random algorithm is used to generate the first N initial points. Then we add those queries and their outputs to an initial dataset to be used for Bayesian optimization. The resulting algorithm is illustrated below.

Given $\boldsymbol{x}$ and an initial perturbation r_1, we have an updated Bayesian optimization algorithm.

Algorithm 5: Bayesian Optimization with N Initial Points

1: run *Algorithm 4: Random* with N maximum iterations
   save queries and outputs to $\mathcal{D}_1$
2: using $\mathcal{D}_1$ as initial dataset, run *Algorithm 2: Bayesian Optimization*

The **Algorithm 2** for Bayesian optimization can be found right before **Chapter 3.3.1**.

3.4.3 BO with Dimensionality Reduction

High dimensionality is a limitation we have discussed in **Chapter 3.3.2**. This issue, in particular, makes the optimization of the acquisition function quickly become intractable in higher dimensions.

We have hypothesized that current machine learning algorithms have low effective effective dimensionality, or are simply too linear [1, 3]. That is, the objective function is only affected by a few of the input features.

Nevertheless, it is worth exploring techniques to mitigate the issue of high dimensionality, so that optimizing the acquisition function will be easier and quicker. The general idea for this attack is to do dimensionality reduction first, then perform Bayesian optimization in the reduced (lower-dimensional) space.

The only step done in the original space will be querying the target classifier. That is, once the acquisition function finds the next point to query in the reduced space, we transform the query back into the original space to input into the target classifier. In theory, it should be easier and quicker to find a global optima as the acquisition function works in a lower dimension space.

For multimodal cases where the input feature space $\boldsymbol{x}$ consists of both discrete and continuous features, it will be difficult to formalize a general framework on reducing its dimensions as it will depend on the dataset. For now, we work with a continuous subspace of the real numbers $X \subset \mathbb{R}^d$, where $\dim(X) = d$.

Formally, if the original input space is $\boldsymbol{x}$ with $\boldsymbol{r} \in X$, let $T : X \to X_{\text{red}}$ be a transformation that maps elements of $\boldsymbol{x}$ into a reduced space X_{red} with $\dim(X) \gg \dim(X_{\text{red}})$. To be able to do Bayesian optimization on the reduced space and feed it back to the target classifier, we also need to have a pseudo-inverse function $T^{-1} : X_{\text{red}} \to X$ to map input back to the original space.

Define $\boldsymbol{r}_{\text{red}} \in X_{\text{red}}$ and $\tilde{\boldsymbol{r}} = T^{-1}(\boldsymbol{r}_{\text{red}})$. The original optimization problem

$$\begin{aligned} \min_{r} \quad & \Omega_{\boldsymbol{x}}(\boldsymbol{r}) \\ \text{s.t.} \quad & \|\boldsymbol{r}\| < \delta \end{aligned}$$

becomes

$$\begin{aligned} \min_{r} \quad & \Omega_{\boldsymbol{x}}(\tilde{\boldsymbol{r}}) \\ \text{s.t.} \quad & \|\tilde{\boldsymbol{r}}\| < \delta \end{aligned}$$

We update **Algorithm 5** to include dimensionality reduction. This is illustrated in **Algorithm 6**.

Algorithm 6: Bayesian Optimization with Dimensionality Reduction

1: do dimensionality reduction to get transformation T
2: run *Algorithm 4: Random* in the reduced space ▷ for initial points
   save queries and outputs to $\mathcal{D}_1$
3: **for** $n = 1, 2, \ldots, M$ **do**
4: select new $\boldsymbol{r}_{n+1}$ by optimizing acquisition function α
   $\boldsymbol{r}_{n+1} = \arg\max_{\boldsymbol{r}} \alpha(\boldsymbol{r}; \mathcal{D}_n)$
5: query target classifier by getting output probability vector of $\boldsymbol{x} + \tilde{\boldsymbol{r}}_{n+1}$
   this gives us the value of $\Omega_{\boldsymbol{x}}(\tilde{\boldsymbol{r}}_{n+1})$
6: augment dataset $\mathcal{D}_{n+1} = \{\mathcal{D}_n, (\boldsymbol{r}_{n+1}, \Omega_{\boldsymbol{x}}(\tilde{\boldsymbol{r}}_{n+1}))\}$
7: update $\mathcal{GP}$ prior and posterior distributions

The acquisition function α and the Gaussian process $\mathcal{GP}$ work in the reduced dimension space X_{red}.

Determining a δ_{red} so that the constraint $\|\boldsymbol{r}_{\text{red}}\| < \delta_{\text{red}}$ in the reduced space translates to $\|T^{-1}(\boldsymbol{r}_{\text{red}})\| < \delta$ in the original space depends on the transformations and norm used. This is difficult even in the simple case where the transformation is linear, so this may become a problem later on.

Recall that the purpose of the box constraint δ was to ensure that the perturbed sample retains its true class label, so performing these transformations runs the risk of generating adversarial samples that are legitimately from another class.

There are multiple ways to do dimension reduction, but most techniques need to identify the most important features of the data. To do this requires a large enough labeled dataset that is representative of the target model's training data. However, gaining this advantage requires us to give up our assumption of a weak adversary.

The first dimensionality reduction technique we consider does not require us to change our assumptions on the adversary. This was implemented by Wang et al. [17] where they implement Bayesian optimization with random embeddings (REMBO). REMBO first draws a random embedding, given by a random Gaussian matrix $\boldsymbol{A}$, and then performs Bayesian optimization in this embedded space.

So in **Algorithm 6**, the transformation T becomes multiplication by $\boldsymbol{A}$ and the acquisition function restricts its search to a bounded region $\mathcal{B}$. However, the choice of $\mathcal{B}$ is very particular, and considering our additional boundary conditions, it becomes very restrictive for the perturbation $\boldsymbol{r}$. Overall, we think the adversarial setting is too restrictive for this implementation at the moment [17].

The other more simple dimensionality reduction technique we consider is principal component analysis (PCA). We will have to relax our assumptions on a weak adversary here, but this dimensionality reduction is simple and easy to implement. PCA works by using eigenvectors to choose the features that contribute to the largest amount of variance.

There is also the concern that some adversarial samples may not have a preimage in the reduced space. Because T maps from a higher dimension space to a lower one, if $x' \in X$ is an adversarial sample then there could be no $\boldsymbol{r}_{\text{red}} \in X_{\text{red}}$ such that $x' = \boldsymbol{x} + T^{-1}(\boldsymbol{r}_{\text{red}})$. However, we will assume that there adversarial perturbations still exist.

Particular details on enforcing the box constraints and number of components used for PCA depends on the data and will be determined as we run experiments in **Chapter 4.3**.

3.4.4 Fast Gradient Sign Method (FGSM)

Goodfellow et al. [1] outlined the fast gradient sign method (FGSM) for generating adversarial examples against neural networks. This is a computationally efficient white-box attack that can be used against any machine learning algorithms that makes use of gradients and weights. FGSM is described in the following paragraph.

"Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the targets associated with $\boldsymbol{x}$ (for machine learning tasks that have targets) and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost used to train the neural network. We can linearize the cost function around the current value of $\boldsymbol{\theta}$, obtaining an optimal max-norm constrained perturbation of $\nu = \epsilon \text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$" [1].

Additionally, the gradient needed can be computed efficiently via backpropagation. Hence, this is a white-box that is efficient to execute if you know the internal weights and architecture of the machine learning algorithm. Moosavi-Dezfooli et al. [21] gives an FGSM algorithm against a general image classifier in the multi-class case that we implement.

The white-box FGSM will be used as an upper benchmark to our black-box Bayesian optimization attack. The random attack is the lower benchmark. We expect Bayesian optimization to perform better than random, but not as good as FGSM.

Chapter 4

Experiments & Results

In this chapter, we discuss our implementation specific to this problem and dataset. We then present and discuss the results of the experiments.

- **Chapter 4.1** lists the remaining implementation and design choices that need to be made specific to our experiment.
- **Chapter 4.2** compares the Bayesian optimization, fast gradient sign, and random attacks. Bayesian optimization has better results than the other attacks.
- **Chapter 4.3** evaluates the Bayesian optimization attack with dimensionality reduction.

4.1 Implementation

In this section we list the software, dataset, machine learning classifiers, and the other design choices used in the following experiments.

Software
This project is implemented on **Python 3.6** with the following libraries.

- **GPyOpt** for constrained Bayesian optimization with Gaussian processes [22],
- **Keras** for convolutional neural networks [23],
- **scikit-learn** for the remaining machine learning algorithms [24], and
- **cleverhans** for implementing the fast gradient sign method against convolutional neural networks [25].

The GPyOpt library allowed us to set up the constrained Bayesian optimization as described in **Chapter 3.3.1**.

4.1.1 Dataset & Classifiers

Dataset
For image classification tasks there are some benchmark datasets that are popular for research such as, MNIST (handwritten digits recognition dataset), CIFAR10 (objects recognition dataset), SVHN (digits recognition dataset), STL10 (objects recognition dataset), and ImageNet1000 (objects recognition dataset). For this research we only use the MNIST dataset as it is widely used as a benchmark in the most prominent research for adversarial machine learning like with Papernot et al. [5, 6, 8]. Additionally, when compared to the

other datasets, MNIST has the least number of features at 784 [19]. In image classification, images have hundreds to thousands of pixel values. It is sensible to start with a lower "high" number of dimensions to test if Bayesian optimization can perform well against these high-dimension input.

The target classifier's input are black and white 28 by 28 pixel images of handwritten digits. These are flattened into vectors with 784 features, with each feature corresponding to a pixel intensity taking normalized values between 0 and 1. The image represents digits from 0 to 9, so the output probability vector has 10 components, one for each label class from 0 to 9.

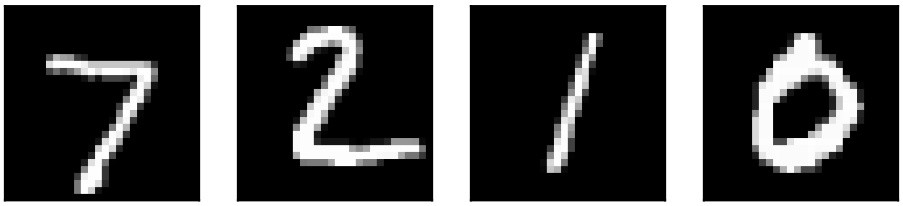

Figure 4.1: MNIST Samples - first few images from the MNIST dataset. Labeled as 7, 2, 1, and 0 from left to right.

Classifiers
For this research, we first work with the most common and effective machine learning algorithms for image classification. We've chosen a convolutional neural network (CNN), logistic regression (LR), and random forest (RF) ensemble.

Convolutional neural networks are a clear choice because they represent the state-of-the-art in image recognition; most well tuned deep neural networks achieving upwards of 99% in test accuracy. The logistic regression and random forest ensemble were both chosen because of their simplicity, effectiveness, and wide use as machine learning algorithms. Other classes of algorithms such as k-means clustering (KM), k-nearest neighbors (KNN), and support vector machines (SVM) were considered, but due to time constraints we decided to first focus on the more common and prominent classifiers for image recognition.

- CNN was implemented using Keras, we detail its architecture below.
- Multinomial LR was implemented using scikit-learn's LogisticRegression class.
- RF ensemble was implemented using scikit-learn's RandomForestClassifier class with 100 estimators and gini impurity as its criteria.

For the convolution neural network, the input is processed by the following layers (in this order): two convolutional layers (32 then 64 kernels of 3 by 3 pixels), a pooling layer (2 by 2 filters), a fully connected hidden layer (128 neurons), and an output softmax layer (10 neurons). The output is a probability vector with 10 components, each corresponding to the digits 0 to 9.

Training & Testing
The MNIST dataset was split into 60,000 training data points and 10,000 testing data points. All classifiers were trained with the same test data and their accuracy evaluated with the same test data. The classifiers label the input image with the class label assigned the largest probability. The CNN achieved a test accuracy of 99.06%, the LR had 92.46% test accuracy,

and the RF had 96.94% test accuracy.

We selected 100 test data points to attack and craft adversarial samples for. These samples were used to evaluate the effectiveness of our attacks. The number of samples was restricted to 100 due to time restrictions; Bayesian optimization is not a quick algorithm.

4.1.2 Metrics & Settings

Design choices involving the number of initial points, number of maximum iterations, and the distance metrics are specific to the type of input data used. We discuss our choices below, specific to the MNIST dataset and image classification task.

Queries
Bayesian optimization does not need many queries to find the optimum. Preliminary experiments show that a little under 100 queries per sample is enough to be effective without slowing down my computing equipment. We decided to use 30 initial points and a maximum of 50 additional iterations as this setting was sufficient.

Note: Since we have a total of 80 queries for the Bayesian optimization attack, we set the maximum number of queries to 80 for a pure random attack for comparison.

Increasing the number of initial points and maximum iterations will improve the performance of the Bayesian optimization, but at the cost of computing time (exponentially $O(n^3)$). Due to time and computational constraints we put that aside for future work.

Distance Metrics
In our discussion of the adversary's secondary goals in **Chapter 3.2.3**, we mentioned how surveying human subjects to classify these modified images is needed to *rigorously* check if perturbations change the true class labels. However, we are unable to do that due to time constraints.

The best substitute for that would be to constrain the magnitude of the change. This is done by using a distance metric to minimize the perturbation $\|\boldsymbol{r}\| < \delta$. Choosing this metric or norm will have some effect on the perturbations generated in image classification tasks [26].

- The L_1 norm $\sum_{i=1}^{n} |r_i|$ encourages localized deformations.
- The L_2 norm $\sqrt{\sum_{i=1}^{n} r_i^2}$ encourages less localized and more smooth deformations.
- The L_∞ norm $\max_i |r_i|$ encourages uniformly spread deformations.

Where r_i is the i-th component of the perturbation vector $\boldsymbol{r}$. Adversarial perturbations generated from the fast gradient sign method (FGSM) create relatively smooth or spread out deformations [1, 21]. There are also adversarial perturbation generated by localized pixel-by-pixel distortions [8]. Hence, for image classification problems, there is a case for each of the norms.

If the classification problem is well studied for adversarial samples, the adversary can make a better guess on which norm or metrics to use. For now, we choose to go with the L_1 norm, in line with our choice of the Matérn 5/2 kernel function, as it does not assume smoothness

in the target function.

Choosing δ
Now to determine δ for the box constraint the adversary can get any of his input samples and apply varying magnitudes of perturbation to see for what values does the picture get too occluded. This is illustrated in **Figure 4.2**.

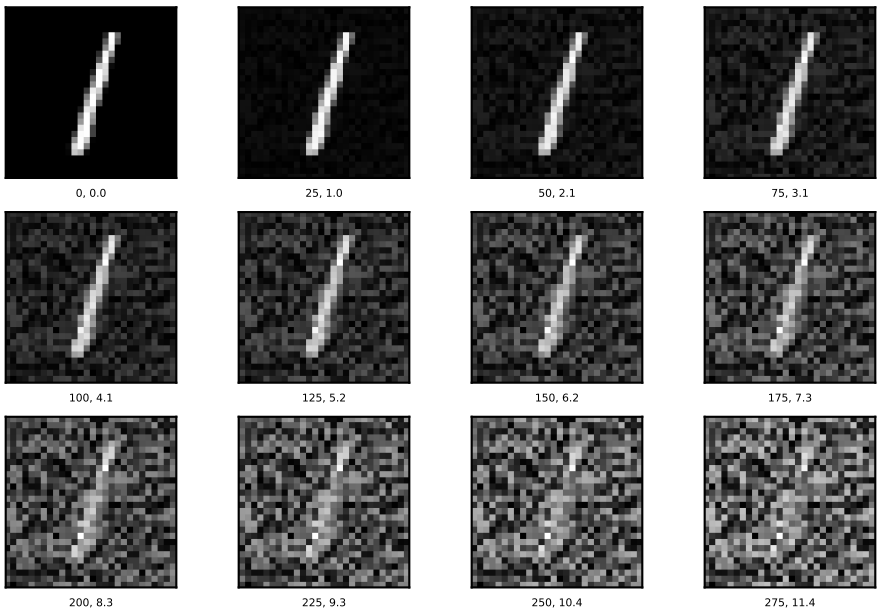

Figure 4.2: Norm Comparison - numbers x, y under each image show the perturbation's L_1 norm (x) and L_2 norm (y).

The noise is barely visible for L_1 norm values from 0 to 50. It begins to show, but only slightly, for 50 to 100. From 100 to 150 the true class label is unchanged, but the noise is fairly obstructive. For higher values of the L_1 norm, the true class label is unidentifiable because of the obstruction. So a good range to test δ would be from 0 to 150.

Aside from the box constraint $\|\boldsymbol{r}\| < \delta$, there is also another boundary condition given by the MNIST dataset. Since we flatten the inputs and normalize their features to the range $[0, 1]$, every input satisfies $\boldsymbol{x} \in [0, 1]^{784}$.

Given an input $\boldsymbol{x}$ with x_i as its i-th component, the i-th component of $\boldsymbol{x} + \boldsymbol{r}$, given by $x_i + r_i$, should be in the range $[0, 1]$. Thus, we have the additional boundary condition for the perturbation $\boldsymbol{r}$: we have $r_i \in [-x_i, 1 - x_i]$ for all $i = 1, \ldots, 784$.

4.2 Bayesian Optimization

We implemented a Bayesian optimization (BO) and a pure random attack on all three classifiers: convolutional neural network (CNN), logistic regression (LR), and random forest (RF) ensemble. Then we implemented the fast gradient sign method (FGSM) against the CNN

and LR. We did not do this for the RF since it did not use gradients.

The 100 input samples for the adversarial attack were chosen from the testing data. The attacks were done for δ values from 0 to 150. Naturally, evasion rates increase for larger δ as it allows larger perturbations.

The best result was that BO successfully evaded these high-accuracy classifiers on a high percentage of the input samples within a reasonable δ. This shows its feasibility as a black-box attack against machine learning classifiers with high-dimensional inputs (MNIST had 784 features). And, surprisingly, it was the most effective attack against the CNN.

CNN was the most resilient of the classifiers, followed by LR. It was moderately easy to evade LR for both BO and FGSM attacks. RF was very easy to evade even for the random attack. This can be seen when in the $\delta = 90$ range, the evasion rates was at most 30% for CNN while it was 99% for LR and 91% for RF. Light noise is visible at the 50 to 100 range for δ. **Table 4.1** summarizes these results.

Classifier-Attack Pairs	$\delta = 5$	$\delta = 20$	$\delta = 40$	$\delta = 60$	$\delta = 90$	$\delta = 120$	$\delta = 150$
CNN-BO	1	2	6	11	30	68	99
CNN-FGSM	1	3	7	14	29	48	62
CNN-Random	0	0	1	1	3	13	32
LR-BO	4	19	77	89	98	99	100
LR-FGSM	12	89	99	99	99	99	100
LR-Random	0	4	23	53	76	81	84
RF-BO	53	78	85	89	91	93	96
RF-Random	28	68	82	86	88	90	92

Table 4.1: All Results - this shows the number of samples evaded out of a 100 for each classifier-attack pair, with its corresponding maximum perturbation δ. Since there were 100 samples, this is equivalent to their percentages.

4.2.1 Attack Comparison

In this part, we compare the various attack algorithms for each classifier. We show that Bayesian optimization (BO) is effective as a black-box attack, outperforming the random attack and having comparable results to the white-box attack (FGSM).

Logistic Regression
For logistic regression (LR), the fast gradient sign method (FGSM) should be the best attack since it makes use of the gradients and LR is a linear convex model. The results verify this. For $\delta = 20$, FGSM has an 89% evasion rate and this goes up to 99% for higher δ.

BO does not do as well as FGSM for $\delta < 40$, but it reaches comparable evasion rates for $\delta \geq 60$. The random attack is used to set a baseline for evasion rates, and it identifies if the target classifier is vulnerable to random noise. BO clearly outperforms the random attack from the start. These are illustrated in **Figure 4.6**.

The results imply that BO is effective in evading a machine learning classifier despite the high dimensionality of the input. This supports the hypothesis that the underlying machine

learning classifier has low effective dimensionality, and supports the claim that BO is viable black-box attack against machine learning algorithms in general.

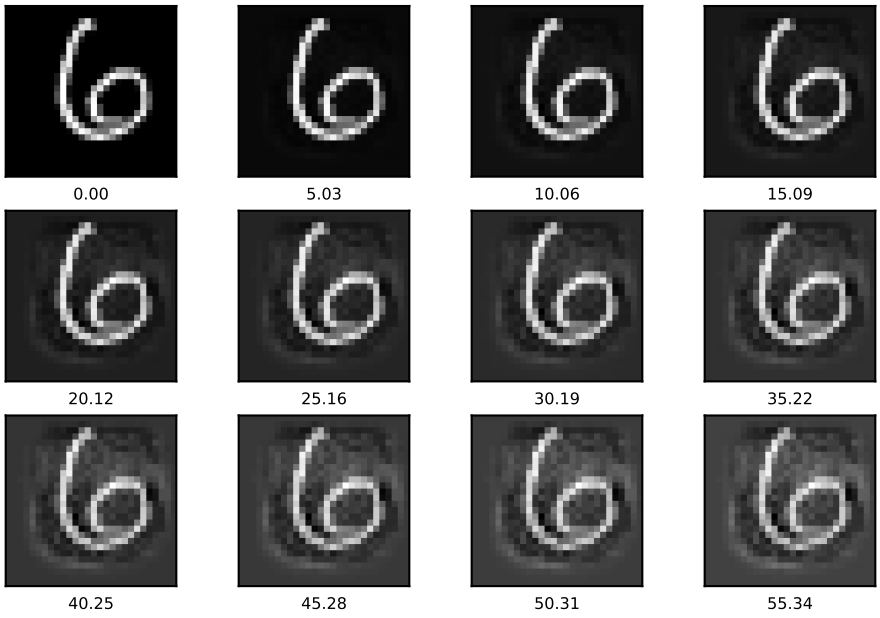

Figure 4.3: LR-FGSM Adversarial Samples - '6' misclassified to '5' in the last 10 images, with the number under each image being the δ of the perturbation.

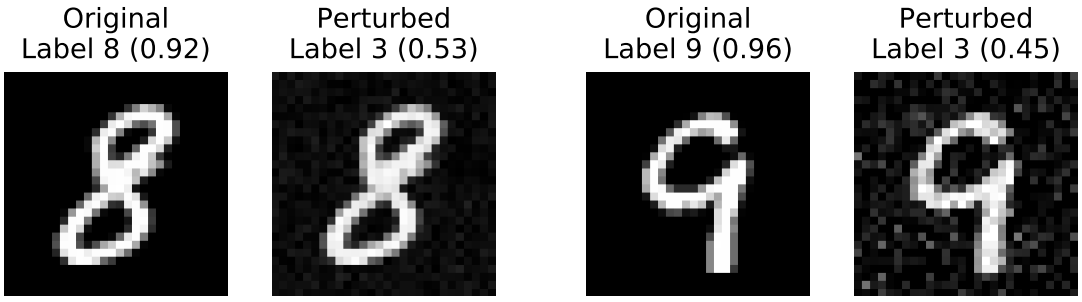

Figure 4.4: LR-BO Adversarial Samples - (Left) '8' misclassified to '3' with $\delta = 35$. (Right) '9' misclassified to '3' with $\delta = 53$.

Comparing the LR adversarial samples between BO and FGSM, the ones generated by BO look less "obvious" than those made by FGSM of the same δ values. This may be due to our use of the L_1 norm, as it encourages localized rather than smooth perturbations like those made by gradient-based methods.

Convolutional Neural Network

The convolutional neural network (CNN) is more resilient than the other classifiers. Even by looking at **Table 4.2**, we can see the evasion rates reach double-digits only when $\delta \geq 60$ and even $\delta \geq 120$ for random.

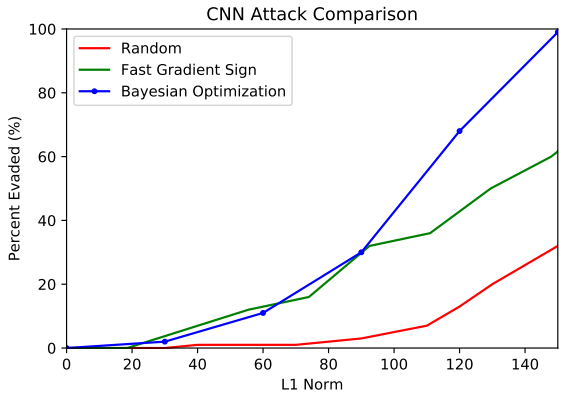

Figure 4.5: CNN Attack Comparison - evasion percentages against CNN for its corresponding δ (maximum L_1 norm).

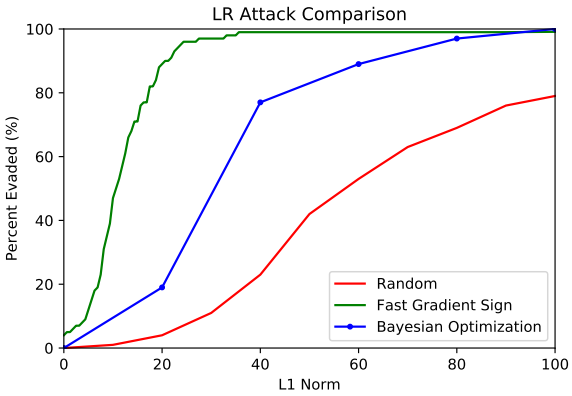

Figure 4.6: LR Attack Comparison - evasion percentages against LR for its corresponding δ (maximum L_1 norm).

Note that L_1 norm of **Figure 4.5** goes from 0 to 150 while that of **Figure 4.6** goes from 0 to 100.

Classifier-Attack Pairs	$\delta = 5$	$\delta = 20$	$\delta = 40$	$\delta = 60$	$\delta = 90$	$\delta = 120$	$\delta = 150$
CNN-BO	1	2	6	11	30	68	99
CNN-FGSM	1	3	7	14	29	48	62
CNN-Random	0	0	1	1	3	13	32
LR-BO	4	19	77	89	98	99	100
LR-FGSM	12	89	99	99	99	99	100
LR-Random	0	4	23	53	76	81	84

Table 4.2: CNN & LR Results - this shows the number of samples evaded (out of 100) for each classifier-attack pair, with its corresponding maximum perturbation δ. Since there were 100 samples, this is equivalent to their evasion percentages.

For $\delta \leq 90$, BO and FGSM have similar or matching evasion rates. It is for large values, $\delta > 90$, where BO begins to outperform FGSM. At $\delta = 120$, we have 68% evasion for BO and 48% evasion for FGSM. By $\delta = 150$, BO is able to evade virtually all samples while FGSM could only get to 62%. Despite this, the random attack did not evade CNN. It only begins to evade at a higher rate at around $\delta > 110$. This suggests that the CNN was extremely resilient to random perturbations. These observations are illustrated in **Figure 4.5**.

We hypothesized that FGSM will outperform BO as it made use of internal knowledge of the CNN where BO had none, but this did not turn out to be the case. It is possible that FGSM was not the best white-box attack against CNNs for image classification, there are other papers that employ other techniques such as pixel-by-pixel distortion. However, these require more a lot more queries [8].

To add to that, FGSM performed better on LR than on CNN because of fundamental differences. The CNN models a more complex and robust function than LR. FGSM may get trapped in local optima when traversing CNN's function, whereas it will not with LR's due to its convexity. Despite the shortcomings of FGSM against CNN, it is still impressive that BO manages to outperform it at higher δ values.

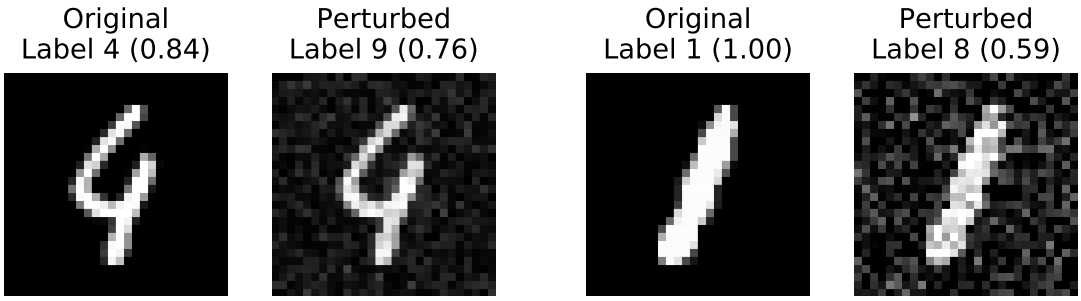

Figure 4.7: CNN-BO Adversarial Samples - (Left) '4' misclassified to '9' with $\delta = 60$. (Right) '1' misclassified to '8' with $\delta = 90$.

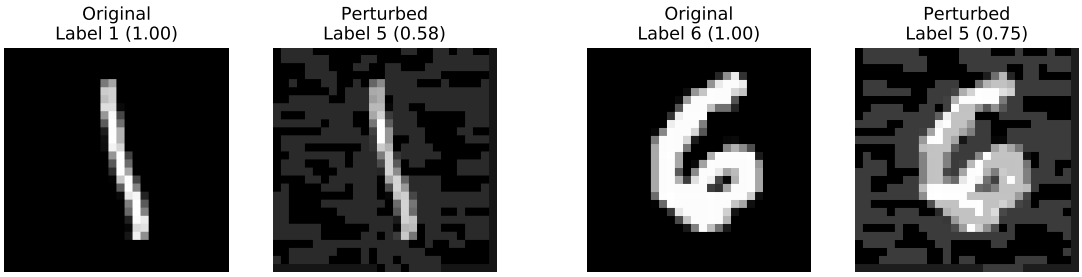

Figure 4.8: CNN-FGSM Adversarial Samples - (Left) '1' misclassified to '5' with $\delta = 74$. (Right) '6' misclassified to '5' with $\delta = 110$.

This implies that even the underlying classifier function of CNN has low effective dimensionality and can be optimized by BO. This is supported by studies that show hyperparameter optimization of deep neural networks to have low effective dimensionality [3]. All these imply that BO is an effective method for black-box evasion attacks against CNNs, a state-of-the-art algorithm.

The CNN adversarial samples of BO resemble salt-and-pepper noise, most likely because of the L_1 norm. The adversarial samples generated by FGSM look more like smooth masks laid over the original image. In all cases, the true class label still remains the same as can be seen in the images.

Random Forest

The random forest (RF) ensemble classifier is a bit different from the other models. The evasion percentage shoots up even at the slightest perturbation. This can be seen for $\delta = 20$, where BO evades it at 78% and random evades it at 68%. The attacks improve further for $\delta = 40$ with BO at 85% evasion and random at 82% evasion. However, the evasion rate seems to plateau for higher δ.

These results, especially the high evasion rate of random noise, suggests a huge vulnerability with the classifier. This vulnerability is surprising as random forests are one of the most successful machine learning algorithms for classification and regression. It will be hard to compare attack methods here since adversarial samples are generated with great ease. We investigate this further in the next subsection.

Classifier-Attack Pairs	$\delta = 5$	$\delta = 20$	$\delta = 40$	$\delta = 60$	$\delta = 90$	$\delta = 120$	$\delta = 150$
RF-BO	53	78	85	89	91	93	96
RF-Random	28	68	82	86	88	90	92

Table 4.3: RF Results - this shows the number of samples evaded out of a 100 for each classifier-attack pair, with its corresponding maximum perturbation δ. Since there were 100 samples, this is equivalent to their percentages.

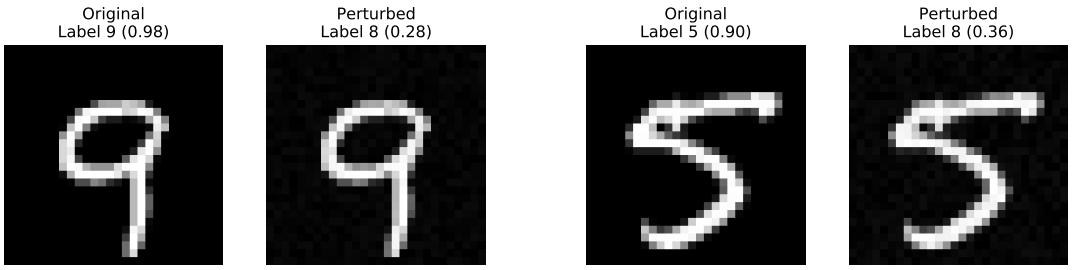

Figure 4.9: RF-BO Adversarial Samples - '9' and '5' are both misclassified to an '8' with $\delta = 20$.

4.2.2 Classifier Analysis

The classifiers' test accuracies were: CNN 99.06%, LR 92.46%, and RF 96.94%. It is no surprise then that CNNs were the most robust to adversarial samples as they are the state-of-the-art in machine learning algorithms for image classification.

CNN has the lowest evasion rate for each attack and LR is almost always second best in lowest evasion. The results of our experiments confirm this and is illustrated in **Figure 4.10**.

However, as mentioned in the previous section, the vulnerability of RF to the slightest perturbation is surprising. The RF achieved a testing accuracy of 96.94%, far better than the 92.46% of LR. Yet it is more susceptible to adversarial samples. We have not yet encountered any literature that explores the vulnerability of ensemble classifiers to random noise.

Ensemble methods are learning algorithms that construct a new model by taking the weighted votes or average of several weaker models. For example, the RF ensemble is an aggregation of decision tree (DT). These methods obtain better predictive performance together versus any single one of their constituent learning models. These are also applicable to both classification and regression tasks.

It is possible that a good number of DTs are vulnerable to any perturbations. Hence, when aggregated into an ensemble model, this weakness gets magnified as each of the DTs would contribute their vulnerability.

The confusion matrices in **Figure 4.11** show that the RF heavily favors classifying towards '8' and then '2' for all perturbed samples. This contributes to why the evasion rate plateaus at about 90% because most adversarial perturbations on '8' do not cause misclassification. This could explain why BO could not achieve a better performance, as the input space that gets labeled '8' is so large, BO would have to explore more samples and a much larger space to find adversarial samples against '8'.

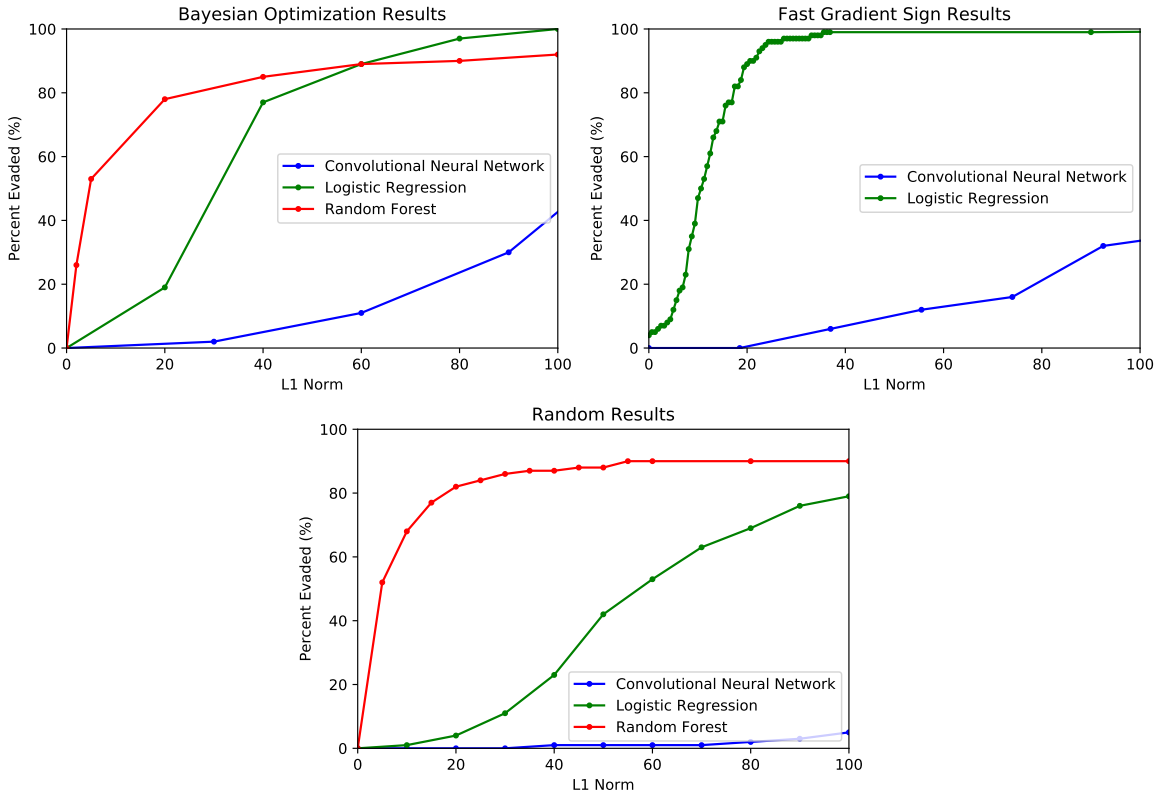

Figure 4.10: Classifier Vulnerability Comparison - each plot represents an attack method and it compares its evasion percentage on each of the classifiers.

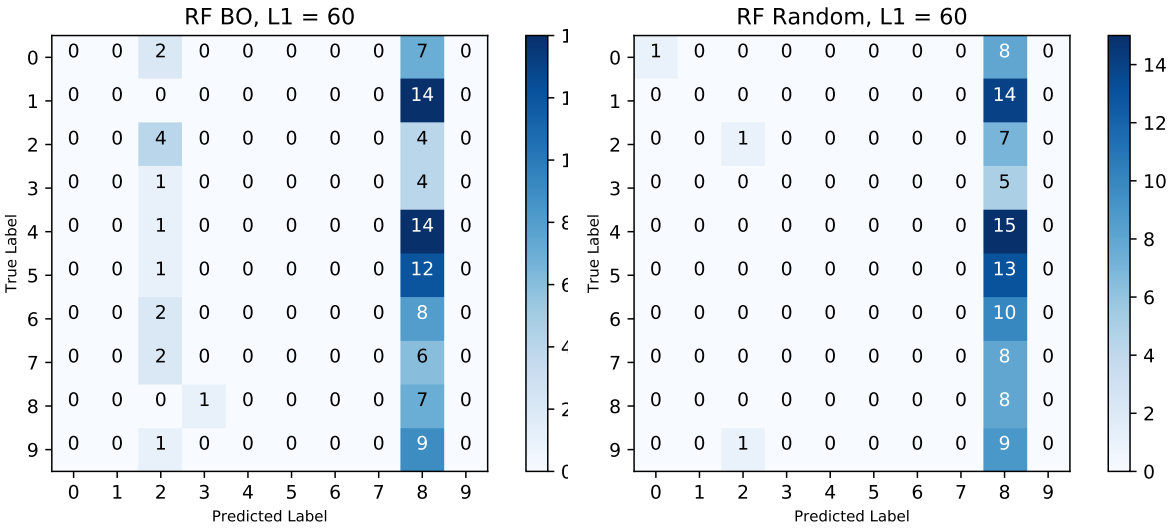

Figure 4.11: RF Confusion Matrices - cell (x, y) represents the number of samples with true class label x classified as y for each attack at the listed δ value.

For high enough δ, we see that the CNN greatly favors classifying towards '8' first, and then '2' and '3' as shown by the BO and random attack matrices in **Figure 4.12**. The adversarial

perturbations of FGSM are more scattered around multiple class labels.

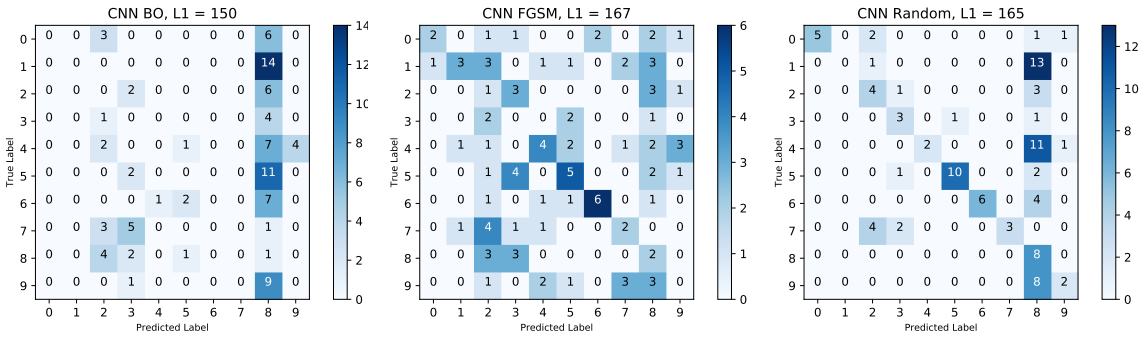

Figure 4.12: CNN Confusion Matrices - cell (x, y) represents the number of samples with true class label x classified as y for each attack at the listed δ value.

In LR, all three attacks expose '2', '3', and '5' as heavily favored classes of LR, with both BO and FGSM also picking up a preference towards '8' for certain class labels such as cell $(5, 8)$. This is illustrated in **Figure 4.13**.

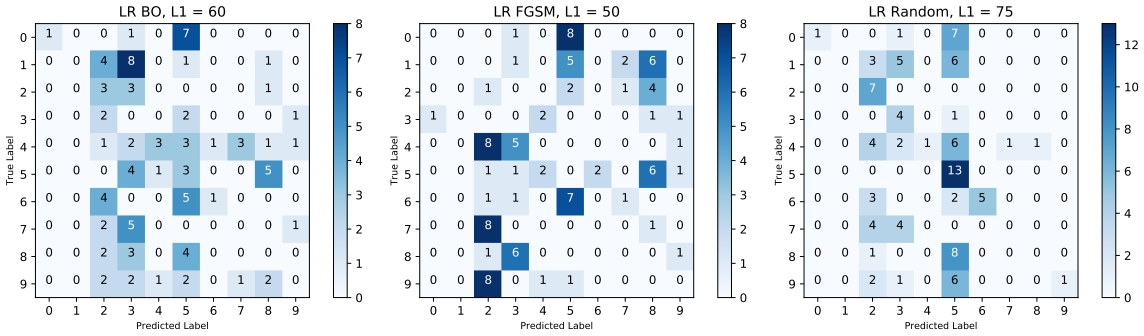

Figure 4.13: LR Confusion Matrices - cell (x, y) represents the number of samples with true class label x classified as y for each attack at the listed δ value.

4.3 Dimensionality Reduction

We implemented a Bayesian optimization (BO) with dimensionality reduction on all three classifiers: convolutional neural network (CNN), logistic regression (LR), and random forest (RF) ensemble. Although we did not strictly measure the running time of these algorithms, this version of BO seemed to run slightly faster than the regular BO.

We used principal component analysis (PCA) on the MNIST training dataset of 60,000 samples to do dimensionality reduction. The number of principal components used ranged was from 15 to 75, testing every 15. The purpose of dimensionality reduction was to speed up Bayesian optimization and hopefully make it more effective, that is why we did not go for more principal components. The results did not vary for different number of components.

One of the primary challenges was adjusting the δ for Bayesian optimization in this reduced space (δ_{red}) because the inverse transformation from the reduced to the original space does

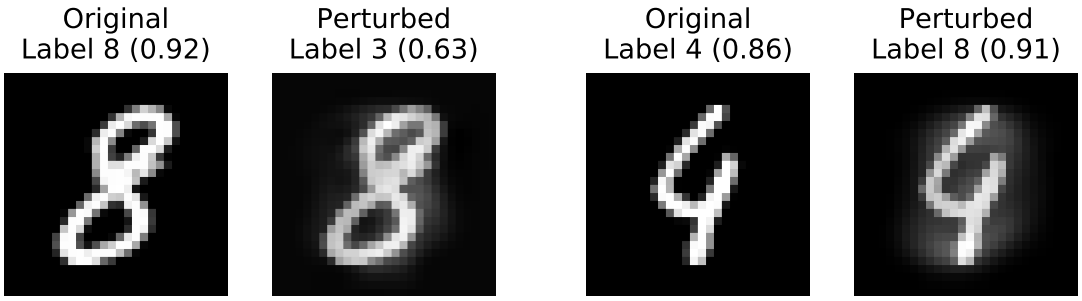

Figure 4.14: LR-BO+PCA Adversarial Samples - (Left) '8' misclassified to '3' with $\delta = 93$. (Right) '4' misclassified to '8' with $\delta = 105$.

not convert the norm linearly. It was difficult to constrain δ_{red} values.

BO+PCA did reasonably well against LR and RF, but it could not evade CNN. Pure BO had better evasion rates than BO. This suggests that pure BO was able to identify certain features to search through and evade the target classifiers. The failure of BO+PCA shows that evading the CNN is not trivial, making the result of pure BO more impressive.

4.3.1 Results

For too low δ_{red} values, BO+PCA fails on all but a few ($<$10%) samples. For too high δ_{red}, BO+PCA creates unrecognizable images. We found that each classifier had a certain range for the δ_{red} where the evasion percentages were high and the perturbations were not obstructive of the image.

In the range where proper evasion occurs, we found that the resulting δ in the original space fell to a narrow range, depending on the classifier. For LR, the δ ranged from 96 to 129 and the best evasion rate was at 72%. For RF, the δ ranged from 100 to 120 and the best evasion rate was at 91%. CNN could not be evaded unless with high δ_{red}, which generated unrecognizable images.

$\delta = 125$	CNN	LR	RF
BO+PCA	3	72	91

Table 4.4: BO+PCA Results - this shows the best evasion rates for BO+PCA at $\delta = 125$. Since there were 100 samples, these numbers are equivalent to the evasion percentages.

The adversarial samples generated looked like the original inputs superimposed with the principal components. This was to be expected since the BO was done on the reduced space that consists of these principal components.

We have discussed in **Chapter 4.2.2** how easy it was to evade the RF classifier, so we can not conclude much from attacks against it. However for the remaining classifiers, the CNN and LR, pure BO achieved much higher evasion at $>$99% on both of the classifiers at $\delta \geq 100$.

Dimensionality reductions took away a lot of features so that BO could be done on a lower-dimensional space. BO+PCA performed worse than BO most likely because PCA took away important features relevant to the target classifier. This is more evident with CNN, which

BO+PCA could not evade in a reasonable way (with legitimate looking adversarial samples).

PCA takes away features with the lowest variance between classes, and it looks primarily for hidden linear correlations. CNN is a more complex algorithm that uses a neural network to create a highly non-linear classifier. It was most likely that PCA took away important features that this BO could have exploited. Our success with pure BO verifies this. BO+PCA however had reasonable success with attacking LR due to the classifier's linear nature.

Combining the results of pure BO and BO+PCA against the CNN, we can infer that BO was able to find and exploit seemingly unimportant (low variance) but vulnerable features of the CNN learned classifier.

4.3.2 Suggestions

PCA may not be the best dimensionality reduction technique that the adversary could have used. Since this is a stronger adversary with access to a representative labeled dataset, he could have done white-box attacks or other forms dimensionality reduction.

Analysis of the images using specialized techniques could have been used. However, these techniques assume certain properties of the target classifier and therefore may only work against certain types of machine learning algorithms. Additionally, these techniques exploit properties specific to the data and image classification tasks. It is in our interest to find a more general framework that can be applied to various classifications tasks.

For a dimensionality reduction technique to utilize the model-agnostic property of BO, the adversary can use something like random embeddings (REMBO) or find a more clever way of reducing the dimensions. The challenge would be to impose the box constraints of the original space so that the adversarial samples generated will retain their true class label. For the image classification problem in particular, the adversary can perhaps choose to only alter a select number of pixels. The challenge with this would be finding the right number of pixels and choosing a good enough combination that allows evasion.

Chapter 5

Conclusion

In this paper, we presented a black-box attack against several machine learning algorithms using Bayesian optimization (BO). We have shown that it is reasonably effective against three different types of classifiers. It was shown to be better than random attacks against all classifiers, and it was shown to have a comparable performance to the white-box FGSM attack against logistic regression.

But more surprisingly, BO outperformed FGSM when attacking the convolutional neural network (CNN) with an evasion rate of 99% for BO and 62% for FGSM at $\delta = 150$. This is important as it shows that Bayesian optimization is an effective technique for generating adversarial samples despite having no information of the target model. It unmasks adversarial samples unknown to the white-box attacks.

The failure of BO with dimensionality reduction against CNN implies that pure BO was able to find and exploit the seemingly unimportant (low variance) but vulnerable features of CNN. All these results support the hypothesis that the underlying machine learning classifier has low effective dimensionality and that BO is a viable black-box attack against machine learning algorithms in general.

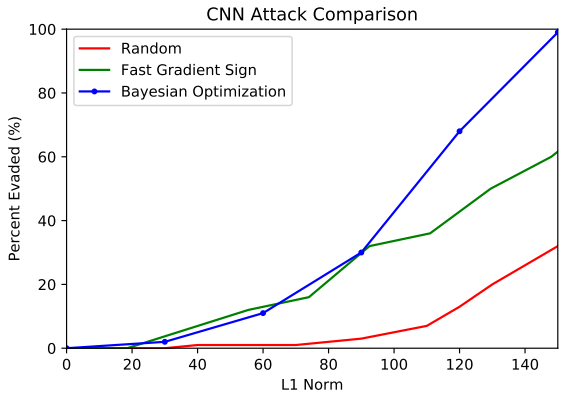

Figure 5.1: CNN Attack Comparison - evasion percentages against CNN for its corresponding δ (maximum L_1 norm).

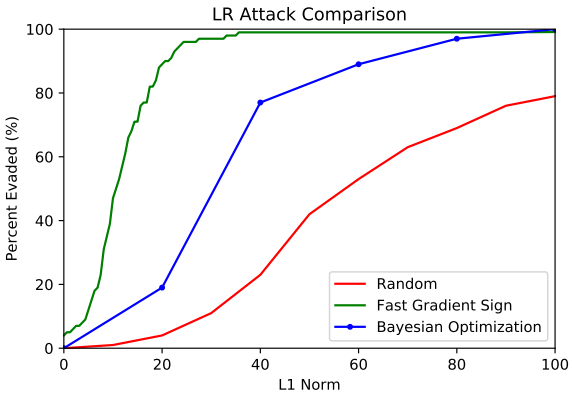

Figure 5.2: LR Attack Comparison - evasion percentages against LR for its corresponding δ (maximum L_1 norm).

These are the primary results comparing Bayesian optimization and fast gradient sign on the more robust convolutional neural network and logistic regression.

These preliminary results are promising and indicate that black-box attacks driven by Bayesian optimization can be effective regardless of the target model or classification task. It has the potential to become a general framework for auditing and performing black-box attacks against existing or new machine learning algorithms and their defenses.

5.1 Suggested Defense

Adversarial samples are difficult to defend against because they require machine learning models to produce good outputs for every possible input. Because of its statistical nature, there is no sure way to defend against Bayesian optimization without getting rid of all the adversarial regions. To do this requires a "perfect" classifier. Researchers acknowledge that it is difficult to make assertions that a specific defense will rule out a set of adversarial samples with absolute certainty because there is no robust theoretical framework for adversarial attack problems [9].

A possible defense against the Bayesian optimization attack would be to mask the output probability vectors. Bayesian optimization needs the continuous, rather than discrete, output to fully utilize and easily compute the Gaussian process. The defender can choose to output only the class label with the highest probability. In applications where the classifier needs to indicate its confidence in the output, it can mask the probability by putting discrete indicators such as 'high confidence', 'low confidence' or by using a discrete scale (e.g. from 1 to 3).

5.2 Future Work

Many interesting ideas and potential improvements for this research were not implemented due to time constraints. We list out some of these ideas as well as some of our shortcomings that can be improved upon in future research.

- Complex Datasets
- Other Classifiers
- Other Classification Tasks
- Ensemble Methods
- Augment Transferability Attacks
- Dimensionality Reduction
- Targeted Evasion

Complex Datasets
For the image classification task, further work can be done on larger more realistic datasets such as the CIFAR or ImageNet datasets. It will be a big result if Bayesian optimization can reliably generate adversarial samples against classifiers for these datasets. We suspect that it can be done because these datasets have a far larger input space, which would mean an exponentially larger search area but also exponentially larger adversarial regions. Attacking classifiers with a larger number of class labels may be easier as well since the confidence of a classifier will be distributed between many more classes.

Other Classifiers
To further test the model-independent property of the Bayesian optimization attack, it can

be deployed against other types of classifiers. This include classifiers we did not cover like decision tree, k-means, k-nearest neighbors, support vector machines, and other types of neural networks.

Other Classification Tasks
The current image classification task only handles with a set of continuous input features. There are classification tasks that are multimodal - they deal with both continuous and discrete input features. It is worth trying, as GPyOpt has an implementation for handling multimodal features. These other classification tasks could include spam filtering or malware detection. Security-oriented applications would be more relevant.

One of the trickier things for these would be to set boundary constraints. The purpose of boundary constraints is to maintain the true class of the original sample. For example, in a malware evasion task, you would start with malware that is correctly classified as one. The boundary constraints should then be set up so that the malware will remain functional after the perturbations have been applied onto it.

Ensemble Methods
As a tangent from Bayesian optimization, one of the interesting results that came up was the vulnerability of random forest ensemble classifiers. These classifiers were surprisingly easy to fool even with small amounts of random noise. It is worth investigating if this was just a fluke or if it is an inherent vulnerability of ensemble classifiers. This has huge implications if it is the latter as ensemble methods are one of the most effective and widely used class of algorithms for classification and regression.

Augment Transferability Attacks
Transferability attacks like those shown in Papernot et al. [6] do not perfectly transfer all their adversarial samples to other machine learning algorithms. We can build a new Bayesian optimization attack using those failed adversarial samples as initial points, rather than using random ones as in this paper's experiment. The idea being that adversarial samples of a classifier may not transfer as adversarial samples to others, but they are most likely in close proximity to one. Using these as initial points, Bayesian optimization can find the adversarial samples for the target classifier much quicker and more efficiently.

Dimensionality Reduction
As discussed in **Chapter 4.3.2**, other ways of dimensionality reduction should be explored. Trying out existing techniques such as linear discriminant analysis or random embeddings, or creating more novel ones would be helpful in making Bayesian optimization more efficient. One can also randomly select a fraction of the features and do adversarial perturbations only on those.

Another idea would be to modify principal component analysis (PCA) to take away the features with the most variance rather than those with the least. It may be these lesser features that need be manipulated in order to evade the classifier, as shown by the results of pure BO compared to BO+PCA. Again, one of the issues with all these methods is setting the box constraints in the reduced space. This may not always translate nicely or linearly when going from the reduced to the original space.

Targeted Evasion

This experiment was mostly concerned with indiscriminate evasion. In the case where the adversary knows which class they want their sample to be misclassified towards, we would need to do targeted evasion. It is worth exploring to see if Bayesian optimization would be as effective or if its capabilities would be restricted and diminished by this.

Bibliography

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

[2] P. McDaniel, N. Papernot, and Z. B. Celik. Machine learning in adversarial settings. *IEEE Security Privacy*, 14(3):68–72, May 2016. ISSN 1540-7993. doi: 10.1109/MSP.2016.51.

[3] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[4] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Marco Melis, Fabio Roli, and Emil C. Lupu. Machine learning under attack. 2017.

[5] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

[6] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[7] Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[8] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

[9] OpenAI. Attacking machine learning with adversarial examples, March 2017. URL `https://blog.openai.com/adversarial-example-research/`.

[10] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

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

[12] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 387–402. Springer, 2013.

[13] Andrea Cassioli and Fabio Schoen. Global optimization of expensive black box problems with a known lower bound. *Journal of Global Optimization*, 57(1):177–190, 2013.

[14] Eric Brochu, Vlad M Cora, and Nando De Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010.

[15] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X.

[16] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[17] Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando de Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

[18] Beilun Wang, Ji Gao, and Yanjun Qi. A theoretical framework for robustness of (deep) classifiers under adversarial noise. *arXiv preprint arXiv:1612.00334*, 2016.

[19] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[21] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[22] The GPyOpt authors. Gpyopt: A bayesian optimization framework in python. `http://github.com/SheffieldML/GPyOpt`, 2016.

[23] François Chollet et al. Keras. `https://github.com/fchollet/keras`, 2015.

[24] 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.

[25] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2016.

[26] Alex Kantchelian, JD Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.