How Wrong Am I? — Studying Adversarial Examples and their Impact on Uncertainty in Gaussian Process Machine Learning Models

Kathrin Grosse∗
CISPA, Saarland University, Saarland Informatics Campus
kathrin.grosse@cispa.saarland

David Pfaff∗
CISPA, Saarland University, Saarland Informatics Campus
pfaff@cs.uni-saarland.de

Michael Thomas Smith
University of Sheffield
m.t.smith@sheffield.ac.uk

Michael Backes
CISPA, Saarland University, Saarland Informatics Campus
backes@cispa.saarland

Abstract—Machine learning models are vulnerable to adversarial examples: minor, in many cases imperceptible, perturbations to classification inputs. Among other suspected causes, adversarial examples exploit ML models that offer no well-defined indication as to how well a particular prediction is supported by training data, yet are forced to confidently extrapolate predictions in areas of high entropy. In contrast, Bayesian ML models, such as Gaussian Processes (GP), inherently model the uncertainty accompanying a prediction in the well-studied framework of Bayesian Inference.

This paper is first to explore adversarial examples and their impact on uncertainty estimates for Gaussian Processes. To this end, we first present three novel attacks on Gaussian Processes: GPJM and GPFGS exploit forward derivatives in GP latent functions, and Latent Space Approximation Networks mimic the latent space representation in unsupervised GP models to facilitate attacks. Further, we show that these new attacks compute adversarial examples that transfer to non-GP classification models, and vice versa. Finally, we show that GP uncertainty estimates not only differ between adversarial examples and benign data, but also between adversarial examples computed by different algorithms.

I. INTRODUCTION

Recent advances in Deep Neural Networks (DNN) have enabled researchers to model complex, non-linear relationships. Of particular interest are classifiers, which take in complex, high-dimensional input and, after passing it through multiple layers of transformations, assign a class. Such classifiers have been used in many application areas over the years, including speech recognition, computer vision, robotics and even automated theorem-proving [25]. It is not surprising that this trend has also carried over towards safety-critical and security-relevant systems, where classifiers are used for an increasingly diverse number of tasks such as malware and intrusion detection, spam classification, self-driving cars and other autonomous systems [38], [2], [29], [45], [10].

However, ML models have been shown to be vulnerable against a broad range of different attacks [3], [26], [11], [52], [45], [20], [5]. Adversarial Examples present the most direct threat to ML classification at test-time: by introducing an almost imperceptible perturbation to a correctly classified sample, an attacker is able to change its predicted class. Among other targets, adversarial examples have been used to craft visually indistinguishable images that are misclassified by state-of-the-art computer vision models [43], [11], [29] and they enable malware to bypass ML-based detection mechanisms without loss of functionality [40], [14], [16], [47].

As an answer to these attacks, defensive mechanisms have been developed [3], [33]. These defenses mostly provide empirical mitigations of adversarial examples, and therefore provide no fundamental proof or guarantee of robustness. Separately, another very recent branch of research is concerned with providing a theoretical foundation of robustness against adversarial examples, e.g. for verification [19], [17]. This work has been mostly focused on DNNs. However, DNNs mostly provide point estimates of parameters and predictions as an output. The theoretical framework underlying DNNs currently lacks the necessary tools to answer questions regarding confidence bounds of predictions. Consequently, it does not provide any information as to whether the data used for training can actually support the predicted output. In fact, it has been hypothesized that adversarial examples are sampled from a distribution separate from the distribution of benign data [13].

In contrast to parametric models such as DNNs, Bayesian non-parametric frameworks, in particular Gaussian Processes (GP) [34], provide information about epistemic and predictive uncertainty. Embedded in the well-studied framework of Bayesian Inference, GPs provide analytical means of answering questions such as ”Is this prediction sufficiently supported by my training data?” . Consequently, researchers have shown renewed interest towards ML domains that take a Bayesian approach and account for uncertainties in spite of their possibly less favorable classification capabilities in benign settings [6].
In this paper, we study Gaussian Process Classification (GPC) and the Gaussian Process Latent Variable Model (GPLVM) in the presence of adversarial examples. In particular, we shed light on how adversarial examples translate to Latent Variable Spaces that Gaussian Processes use to model complex, non-linear relationships between possibly interdependent features. We also investigate the transferability [31, 34] between different models, i.e. how well adversarial examples from classical DNN attacks translate into attacks for GPC and GPLVM, and vice versa. We conclude that adversarial example attacks on Gaussian Processes are effective and translate well into other domains in a gray box setting. Moreover, we empirically evaluate the effects of such adversarial examples on confidence estimates and find evidence that such estimates can be used to distinguish the distributions of adversarial examples and benign data points.

In detail, our contributions are as follows:

- We propose two new attacks to compute adversarial examples for GPC: GPFGS and GPJM. Both attacks are based on the well-studied FGSM [11] and JSMA [32] attacks. We show formally how to compute the necessary forward derivative for GPC.
- We propose Latent Space Approximation Networks (LSAN) as a means of modeling the latent space of non-parametric GP models using DNNs. Thereby, LSAN allows us to directly compute attacks on GPLVM-based classifiers and other kernelized classifiers.
- We study the transferability of standard attacks, LSAN, GPFGS and GPJM in a grey-box setting, where the attacker has access to the training data, but not the ML model itself. We observe that the attacks transfer between all different models of attacks and vice versa.
- We empirically study how adversarial examples affect the uncertainty estimates of GPs. We observe that the uncertainty estimates differ between malicious and benign data. We also take a first step towards leveraging these findings for defenses.

II. RESEARCH QUESTIONS

In this section, we present the research questions (RQ) we investigate in this paper. The overarching goal of stating and subsequently, answering these research questions is to deepen the understanding of how adversarial examples translate to a Bayesian Inference setting using Gaussian Processes as a specimen. We provide the background to deeply understand these questions in the followings sections. The research questions are divided into three main areas of interest: test-time attacks on GP classification, test-time attacks on GPLVM based classification and the impact of adversarial examples on uncertainty in a GP.

A. Attacks on GP Classification

The first research question is concerned with the susceptibility of GP classification to adversarial examples in a worst-case scenario.

RQ 1: Is an adversary with full knowledge of both the GPC model and the underlying training dataset able to craft adversarial examples for a given GP classifier?

We answer this question by introducing two novel attacks on GP classification which we introduce in more detail in Section IV. Subsequently, we want to analyze how well these examples transfer to other ML models trained on the same dataset, and vice versa.

RQ 2: Do adversarial examples crafted on a GPC transfer to other classifiers for the same dataset?

RQ 3: Do adversarial examples crafted on other classifiers transfer to GPC trained on the same dataset?

B. Latent Space Approximation Network attacks on GPLVM Classification

Similar to RQ 1,3 we are interested in how well an attacker is able to craft adversarial examples on GPLVM Classification. As a part of this attack, we introduce Latent Space Approximation Networks, which try to approximate the Latent Variable Model of GPLVM based classification. In addition to evaluating the effectiveness of an attack we are also interested in how well this approximation simulates its target’s behavior. Our target metric for this comparison is the classifiers’ respective accuracy on a benign test dataset.

RQ 4: Does LSAN produce a classifier with accuracy significantly better than random choice?

Furthermore, we evaluate how effective the attacks crafted on LSAN.

RQ 5: Does LSAN allow us to craft attacks on the approximated GPLVM+SVM/linear SVM classifier that decrease accuracy more than other attacks?

In this setting, we distinguish between the data as represented by GPLVM and the kernel used by other ML models. While LSAN were created to craft attacks on GPLVM classification, they are are a generic approach an could potentially be used to craft attacks for any other kernelized method or classifier. Another interesting question is to what extent LSANs inherit the emulated model’s robustness to adversarial examples (if any) given that LSAN is a standard DNN. To obtain interpretable results, we consider three comparisons: DNN and the approximated model (as a baseline), the approximated model and LSAN, and finally LSAN and DNN.

RQ 6: Given a latent space model, its corresponding LSAN and a DNN with the same architecture as the LSAN:

a) When exposed to adversarial examples, will a latent space model and a DNN classify these examples with different accuracies?

b) When exposed to adversarial examples, will a latent space model and its corresponding LSAN classify these examples with different accuracies?

c) When exposed to adversarial examples, will an LSAN and a DNN classify these examples with different accuracies?
C. Influence of Adversarial Examples on Uncertainty

As briefly noted in Section I in addition to providing point probability estimates for hypothesis classes, GP classification also provides information about the uncertainty of its posterior prediction. The unsupervised counterpart of GPC, GPLVM, also provides uncertainty estimates over its latent representation. A common belief within the research community is that estimates of the predictions certainty could potentially be exploited to distinguish adversarial examples.

RQ 7: Do uncertainty estimates for GP classification differ between adversarial examples and benign samples?

RQ 8: Do uncertainty estimates for GPLVM latent space mapping differ between adversarial examples and benign samples?

III. BACKGROUND

In this section, we briefly review ML classification, Deep Neural Networks (DNN) and Support Vector Machines (SVM) before providing an introduction to Gaussian Processes Classification (GPC) and Gaussian Process Latent Variable Model (GPLVM) based classification. We also formally define the threat model by providing a definition of the considered adversaries and attacks.

A. Classification

In classification, we consider a dataset \( \{X_{tr}, Y_{tr}, X_t, Y_t\} \). The goal is to train a classifier \( F(\cdot, \theta) \) by adapting the parameters \( \theta \) based on the training data \( \{X_{tr}, Y_{tr}\} \) such that \( F(X_t, \theta) \approx Y_t \), i.e. \( F \) correctly predicts the label \( Y_t \) of before unseen test data \( X_t \).

B. Test-Time Attacks and Adversarial Examples

Given a trained classifier \( F(\cdot, \theta) \), test-time attacks try to find a small perturbation \( \delta \) for a test sample \( x \in X_t \) such that

\[
\min \delta : F(x, \theta) \neq F(x + \delta, \theta)
\]

i.e., the sample \( x' = x + \delta \) is classified as a different class than the original input. The sample \( x' \) is then called an adversarial example.

To compute the perturbation \( \delta \), we compute the derivative of loss w.r.t. the input, as introduced by Szegedy et al. [42]. Goodfellow et al. [11] refined this, introducing the Fast Gradient Sign Method (FGSM). This attack introduces a global perturbation dependent on a given \( \epsilon \) and the previously mentioned derivative. Another attack, introduced by Papernot et al. [32], is based on the derivative of the trained model w.r.t. the inputs near a particular sample. This approach is called the Jacobian Based Saliency Map Approach (JSMA). Many attacks have been introduced based on this methodology [4, 30, 7]. Although superficially different, Engelbrecht [9] showed that two methods are actually equivalent under some conditions (though differing in the used norm).

Additionally, there exist other methods to compute adversarial examples based on genetic programming [46], mimicry [27, 4] or generative networks [16]. Finally, adversarial examples are not limited to DNNs and there exist variants specific to other ML models [31, 28, 15].

C. Threat Model

We distinguish different threat models based on adversarial knowledge and adversarial capabilities.

Adversarial Knowledge. In the white-box setting, the adversary has unrestricted access to the model, its parameters and its training data. A black-box adversary has access to neither the model nor the training data. In contrast, the grey-box adversary has no knowledge of the model and its parameters, but is still granted access to the training data. This is equivalent to giving the attacker access to a query-able oracle, leaving the attacker oblivious with regards to the ML algorithm applied while still providing her unrestricted access to classification results. Although security in a white-box setting is a more desirable security-property, we investigate grey-box setting as a more tractable and realistic goal for ML models to achieve in an adversarial environment: the trained algorithm is usually protected from unauthorized access, but its training data might be crowd-sourced or even publicly accessible.

Adversarial Capabilities. In the adversarial example setting, we seek to constraint the adversarial capabilities by upper-bounding the amount of perturbation \( \delta \) an attacker can introduce to a sample without the change becoming noticeable to an oracle. This requires one to define appropriate distance metrics. These metrics relate to the type of data: Real valued, normalized data contains features in the interval between \([0, 1.0]\), whereas binary data only provides two values for each feature, 1 and 0. Typical metrics for attacks are the \(L_0\), \(L_2\) and \(L_\infty\) norm. The \(L_2\) and \(L_\infty\) norms are well suited to image or real-valued data, as they measure introduce small changes in multiple features. For binary data, it is impossible to control the amount of perturbation of individual features: values are either flipped or not.

Transferability and Attacker Capabilities in this paper. In this work, we investigate transferability, i.e. the ability of adversarial examples from the same dataset to generalize to different ML classifiers trained on the same dataset. We therefore consider a grey-box adversary when evaluating RQs relating to transferability to exclude other factors that might influence the models and thus affect transferability. We further consider a white-box adversary to evaluate RQs relating to the vulnerability of GPC and GPLVM.

Finally, we use \(L_0\) attacks on binary data and \(L_2\) or \(L_\infty\) attacks on real-valued data.

D. (Deep) Neural Networks

A neural network \( F \) is composed of layers, where the output of one layer is fed as input into the next layer. The weights of layer \( i \) weight each particular input \( x_{i-1} \) according to a weight \( w_i \) and a bias \( b_i \) and then apply an activation function \( f_i \), usually a Rectified Linear Unit (ReLU). We consider \( \theta \) to be the set of all weights and biases in the network. A Deep neural network (DNN) also contains several intermediate
layers, which are referred to as hidden layers. The optimization problem arising when such a deep network is trained is non-convex, hence neural networks are trained using stochastic gradient descent.

\section*{E. Support Vector Machines}

In contrast to DNNs, Support Vector Machines (SVM) form a convex problem with a unique solution. Given the training data, the goal is to compute a hyperplane separating the data. To achieve a nonlinear decision boundary, a SVM can be kernelized. Since the RBF kernel is similar to the covariance function in GP settings, as we will see now, we investigate RBF SVM in this work. To be able to tell apart SVM and RBF effects, we further also study the linear SVM. Another reason to study SVM is that, as we will see now, GPLVM only yields a latent space, and not a classification output. We thus have to apply a classifier on top, and need the original classifier without the preprocessing as a baseline. When employing SVM on top of GPLVM for classification we pick the kernel according to performance.

\section*{F. Gaussian Processes}

We consider two applications of Gaussian Processes. First, we use GP directly for classification (GPC). Second, we describe the Gaussian Process Latent Variable Model (GPLVM), which is a mapping from the data to a latent space. Subsequently, GPLVM is used as a pre-processor to train a SVM classifier.

**GP Classification.** We introduce here GPC [34] for two classes using the Laplace approximation. The goal is to predict the labels $Y_t$ for the test data points $X_t$ accurately. We first consider regression, and assume that the data is produced by a GP and can be represented using a covariance function $k$:

$$y_t = K_{tt} - K_{tr} K_{rr}^{-1} y_t,$$  \quad (3)

which is at the same time the mean of our latent function $f_x$. We will not go into details here on how to optimize the parameters of the covariance function $k$, but instead move from regression to classification. Since our labels $y_t$ are not real valued, but class labels, we ‘squash’ this output using a link function $\sigma(\cdot)$ such that the output varies only between the two classes (cf. Figure 1c). For classification, the optimization can be simplified using the previously stated Laplace approximation. At this point, we want to refer the interested readers to Rasmussen et al. [34].

In addition to the mean, as explained above, we are also able to compute the variance. This allows us to obtain the uncertainty for GPC, and will be used later in this work.

**GP Latent Variable Model.** GPC learns based on labels. This introduces an implicit bias: we assume for example that the number of labels is finite and fixed. Furthermore, the curse of dimensionality affects classification. This curse affects distance measure on data points. In higher dimensions, the ratio between nearest and furthest point approximates one. All data points are thus uniformly distant, impeding classification: it becomes harder to compute a separating boundary. Consequently, we are interested in finding a lower dimensional representation of our data.

These two issues are taken into account when using GPLVM. Its latent space is a lower-dimensional representation of the data where labels are ignored. A particular property of this latent space is further that it allows for nonlinear connections in the feature space to be represented. Additionally, GPLVM models uncertainty estimates, analogously to GPC.

\section*{IV. Methodology}

In this section, we first introduce two direct attacks on GP classification. Afterwards, we introduce Latent Space Approximation Networks (LSAN) to approximate GPLVM-based classification. As we will see, these attacks can be leveraged to attack other kernelized methods.

\subsection*{A. Attacks on GPC}

As for all classifiers, to produce an adversarial example we need to find the gradient in the output with respect to the input dimensions. We consider the chain of gradients for the output $\sigma_x = \sigma(f_x)$, and (test) input $x^* \in X_t$:

$$\frac{\partial \sigma_x}{\partial x^*} = \frac{\partial \sigma_x}{\partial f_x} \times \frac{\partial f_x}{\partial k} \times \frac{\partial k}{\partial x^*}$$ \quad (4)
where \( f_* \) and \( k \) are the associated latent function and covariance function, respectively.

Note that for our attack, we are only concerned with the relative order of the gradients, we are not interested in the actual values. Unfortunately, \( \sigma_* \) does not vary monotonically with \( f_* \). For example, consider that if \( f_* \) increases slightly, thus becoming more positive, the variance of \( f_* \) would increase massively as well. However, through the link function the mean of \( \sigma_* \) would at the same move downwards, approaching 0.5.

We use the gradient of \( f_* \) instead a numerical approximation to \( \sigma_* \). The reasoning is as follows: since we are in a setting of binary classification, we are only interested in moving the prediction across the 0.5 boundary. No change in variance can cause this, instead a change in the mean of \( f_* \) is required. Indeed one can note that the gradient of \( f_* \) is an upper bound on the gradient of \( \sigma_* \). This is due to the fact that the link function ‘squares’ the input, however preserving the changes. The fastest we can get from one probability threshold \( p_t \) to its opposite \( 1 - p_t \) is when there is no variance. So finding the gradient of \( f_* \) is sufficient.

Let us first rewrite the expected value of \( f_* \) in Equation (3) given a single test point \( x^* \):

\[
E[f_*] = k(x^*)^T K_{tr}^{-1} y_{tr}
\]  
\[\text{(5)}\]

From here, we can easily move on to the first part of the gradient,

\[
\frac{\partial f_*}{\partial k} = K_{tr}^{-1} y_{tr}
\]  
\[\text{(6)}\]

where the terms are both constant w.r.t. the test input \( x^* \). The gradient of the kernel w.r.t. the inputs depends on the particular kernel that is applied. In our case, for the RBF kernel, between training point \( x \in X_{tr} \) and test point \( x^* \), the gradient can be expressed as

\[
\frac{\partial k(x^*, x)}{\partial x_i^*} = \frac{1}{l^2}(x_i - x_i^*)k(x^*, x)
\]  
\[\text{(7)}\]

where \( x_i \) and \( x_i^* \) each denote feature or position \( i \) of the corresponding vector or data point and \( l \) denotes the length-scale parameter of the kernel. Using the assumption above the gradient of the output \( \sigma \) with respect to the inputs is proportional to the product of Equation (6) and Equation (7).

Finally, we introduce the attacks, conceptually related to FGSM and JSMA. The first one introduces a global change based on the sign of the gradients (of a single sample \( x \) and the classifier \( f_* \)) and some \( \epsilon \). We depict it in Algorithm 1 and, due to its similarity with FGSM we call it GPFGS. For both attacks we assume the kernel to be RBF (and thus drop the explicit notation of the kernel gradients). We also compute the gradient as the product of Equation (6) and Equation (7) in both algorithms.

In Algorithm 2 we compute the gradients and add or remove a feature depending on the strongest gradient, analogous to JSMA. Consequently, we call this attack GPJM. In this algorithm, we keep track of the modified features, to avoid changing a feature twice. We also allow changes in both directions of the gradients, as visible in line 10. We iteratively change features until a sample is misclassified, or too many features are changed (more than \( t \)).

B. Attacks on GPLVM using Latent Space Approximation Networks

The GPLVM mapping is, similar to kernels, hard to invert. The change in the original data for a corresponding change in latent space is usually hard to determine. We thus try to approximate these methods with DNN, which are easier to invert: there is a wide range of known attacks. The challenge is therefore to produce a substitute Latent Space Approximation Network (LSAN), which is as similar as possible to the original latent space kernel.

The rough idea is depicted in Figure 2. We approximate the latent space and the classifier by using a DNN. To achieve this, we train the second last layer to mimic the latent space, and treat the last layer as a linear classifier (trained on the latent space as input). If this classifier happens to be nonlinear, we add another intermediate layer to achieve nonlinearity. To apply the known attacks for neural networks, we combine the two parts that are trained separately. As visible in Figure 2 we do this by feeding the output of the latent network directly to the classifier network.

Finally, a slight adaption to this scheme also enables us to approximate kernelized SVMs. The difficulty in this setting is that the kernel is a distance function between two points, and thus the latent space is more implicit than in GPLVM. We

Algorithm 1 Given an input sample \( x \) and the latent function \( f_* \), we compute an adversarial example, \( x^* \) using GPFGS. \( \epsilon \) denotes the perturbation introduced for each feature.

Require: \( x, f_*, \epsilon \)
1: \( x^* \leftarrow x^* + \epsilon \times \text{sign}(\nabla f_*) \)
2: return \( x^* \)

Algorithm 2 Given an input sample \( x \) and the latent function \( f_* \), we compute an adversarial example, \( x^* \) using GPJM. \( t \) denotes the maximal number of features we are allowed to change.

Require: \( x, f_*, \sigma_*, t \)
1: \( x^* \leftarrow x \)
2: changed ← [ ]
3: while \( \sigma_*(x) > 0.5 \) equals \( \sigma_*(x^*) > 0.5 \) do
4:     if length(changed) > \( t \) then
5:         return Failure
6:     end if
7:     grads ← \( \nabla f_* \)
8:     grads[changed] ← 0.0
9:     index ← \text{max} (abs(grads))
10:    \( x^* \) [index] ← \( 1.0 \times \text{sign}(\text{grads}) \) [index]
11:    changed.append(index)
12: end while
13: return \( x^* \)
hence train the output layer of the latent space on the kernel matrix given some fixed points from the dataset. More precisely, the output neurons 1...n are trained on the particular distance of some data points 1...n selected from $X_{tr}$.

We can then proceed by using any of the DNN-specific attacks on the LSAN to produce adversarial examples for the original classifiers.

V. EXPERIMENTAL SETUP

Before we evaluate the research questions posed in Section II, we briefly describe the setting of our evaluation. We start with the datasets on which the different ML models where evaluated, and move then to the ML models used and their respective implementations. Before diving into the evaluation, we also briefly comment on convergence and accuracies of all models.

A. Data

We seek to evaluate adversarial examples in two security contexts: Malware classification and Spam detection. In addition to this, we want to evaluate all mechanisms in a commonly used ML benchmark for classification tasks. We carefully selected these datasets to account for the diversity in data types: ML models face across different application domains: individual data features may contain binary (e.g. static code features), integer (e.g. pixel values) or real-valued (e.g. frequencies) data. Whole datasets may contain widely different amounts of features and with different feature sparsity.

**Hidost.** Our Malware dataset consists of the PDF Malware data of the Hidost Toolset project [41] The dataset is composed of 439,563 PDF Malware samples, of which 407,037 are labeled as benign and 32,567 as malicious. Datapoints consist of 1223 binary features and individual feature vectors are likely to be sparse. We split it in 95% training and 5%. This still leaves us with more than 20,000 test data points to craft adversarial examples, where many attacks are very time consuming to compute.

**Spam.** The second security-relevant dataset is an email Spam dataset [23]. It contains 4,601 samples. Each sample captures 57 features, of which 54 are continuous and represent word frequencies or character frequencies. The three remaining integer features contain capital run length information. This dataset is slightly imbalanced: roughly 40% of the samples are classified as Spam, the remainder as benign emails. We split this dataset randomly and use 30% as test data.

**MNIST.** Finally, we use the MNIST benchmark dataset [22] to select two additional, binary task sub-datasets. It consist of roughly 60,000, 28 × 28 pixels, black and white images of handwritten single digits. There are 50,000 training and 10,000 test samples, for each of the ten classes roughly the same number. We select two binary tasks: 1 versus 9 and 3 versus 8 (denoted as MNIST91 and MNIST38 respectively). We do this in an effort to study two different tasks on the same underlying data representation, i.e. the same number and range of features, yet with different distributions to learn.

B. Models

We investigate transferability across multiple ML models derived by different algorithms. In some cases, dataset-specific requirements have to be met for classification to succeed.

**GPLVM.** We train GPLVM generally using 6 latent dimensions with the exception of the Spam dataset, where more dimensions (32) are needed for good performance. We further use SVM on top of GPLVM to produce the classification results, a linear SVM for the MNIST91 tasks and an RBF-kernel SVM for all other tasks.

**LSAN.** We distinguish between LSAN approximating GPLVM (GPDNN), linear SVM (linDNN) and RBF SVM (rbfDNN). All of them contain two hidden layers with half as many neurons as the datasets’ respective features. The layer trained on latent space encompasses 30 neurons for the SVM networks and 6 neurons for GPDNN, except for the Spam dataset, where we model 32 latent variables. From this latent space, we train a single layer for classification, with the exception in GPDNN in the cases where an RBF-SVM is trained on top: here we add a hidden layer of 2 neurons.

**DNN.** Our simple DNN accommodates two hidden layers, each containing half as many neurons as the dataset has features, and ReLU activation functions. For the transferability experiments, the DNN has the same architecture as GPDNN.

**SVM.** We study a linear SVM and a SVM with an RBF kernel. They are optimized using squared hinge loss. We further set the penalty term to 1.0. For the RBF kernel, the $\gamma$ parameter is set to 1 divided by the number of features.
C. Implementation and third party libraries

We implement our experiments in Python using the following specialized libraries: Tensorflow [11] for DNNs, Scipy [18] for SVM and GPy [21] for GPLVM and GPC. We rely on the implementation of the JSMA and FGSM attack from the library Cleverhans version 1.0.0 [12]. We use the code provided by Carlini and Wagner for their attacks [3]. We implement the linear SVM attack (introduced in [31]) and the GAttacks (based on GPy) ourselves.

D. Accuracy and Convergence of Target Models

Before we start investigating the actual transferability task and the adversarial example crafting, we want to comment on the achieved accuracies on the previously defined datasets. Since we encountered some difficulty training the LSANs, we comment on this as well.

Convergence. LSAN networks need to be trained for 7000 iterations, much longer than a conventional DNN. Also its parameters, such as the initial weights, learning rate and bias initialization need to be carefully selected to have the two networks converge.

For RBF-DNN, the LSAN approximating RBF-kernel SVMs, we could find no initial parameters to guarantee convergence globally. The same holds for the LSAN and a linear kernel on MNIST38 and the Malware data. In two settings, RBF kernel on MNIST91 and linear kernel on MNIST38, we could not find parameters working with two different random seeds (one for white box or direct crafting, one for evaluation in grey box). We thus excluded those networks from the experiments, and marked them in Table I.

Accuracy. The test accuracies of all classifiers are depicted in Table I. We only want to compare ML models which achieve similar accuracies, as not doing so might influence the results in an unpredictable manner. In cases where specific models fail to achieve similar accuracy we exclude the outliers.

For the Spam dataset, the highest accuracy is achieved by DNN (94.2%), other algorithms (GP-DNN, GPLVM, GPC, linSVM) vary between 90.1% and 92.7%. On the Malware dataset, all classifiers perform better than 99.2%, where LSANs trained on kernels do not converge. GPDNN achieves the worst accuracy with 98.2%. On MNIST91, we also observe all algorithms to perform better than 99.2%, with the exception of the LSAN trained on a linear kernel (only 98.9). For MNIST38, the range of accuracies of the resulting classifiers is more diverse. Most classifiers (DNN, GPLVM, GPC and RBFSVM) achieve 97.4% to 98.6%. Linear SVM achieves 96.8%, GP-DNN and GPC 94.4%.

We exclude all models with low baseline performance, such as rbfDNN (all settings) and RBF SVM on Spam. LinDNN is excluded on all datasets except MNIST91, where however we use it for crafting on the Spam data.

VI. Evaluation

In this section, we answer all posed research questions given the previously introduced setting. We start with white box setting to provide more background, and then move towards the grey box setting.

In the white box setting, we aim at answering the first RQ. This question, RQ 1, inquires whether we are able to craft adversarial examples on GPC in a white box setting.

We then turn to the grey box setting. Here, we start answering the questions concerning LSAN, RQ 4–RQ 6. We then describe our results concerning GPC and transferability to and from GPC, thus RQ 2 and RQ 3. Finally, we focus on the uncertainty estimates of GP models, and deal with RQ 7 and RQ 8.

A. White Box

Before we start evaluating the experiments, we briefly want to remark on the attacks used on each dataset. In Section III-C we argued which norms are feasible for which data. Since we propose new attacks, however, we ignore these restrictions for the newly introduced attacks to evaluate them without any preconceived bias regarding their usefulness. We also kept other attacks that are not too time consuming in computation, e.g. FGSM, as a comparison.

For each attack, we briefly state how many adversarial examples could be computed using the attack on the benign test data. Additionally, we investigate there average amount of perturbation necessary for misclassification. This measurement refers to either the perturbation computed by the attack (as JSMA, GPJM, $L_\infty$) or the perturbation introduced globally ($\epsilon$, for FGSM, GPFGS and the attack on linear SVMs).

JSMA. The complete results can be found in Table II (correctly classified adversarial examples) and Table III (perturbations needed). On both MNIST task, JSMA is able to produce adversarial examples for all given test samples. This is not the case in all scenarios: JSMA can only produce adversarial examples causing misclassification for roughly half of the samples in the test dataset when targeting LSAN approximating a linear kernel on MNIST31. We also observe, for both LSAN on MNIST91, a higher perturbation needed to fool the respective classifiers. For the other MNIST task we do not observe a noticable difference.
For LSAN (GPDNN), the accuracy falls from effective. Compared to the other attacks, this attack is not very the worst case of this attack is roughly the same as a random test sample. We also measure the perturbations and depict them algorithms succeed in crafting adversarial examples for every classifiers and datasets. Across all datasets, we observe that the grey-box setting later on, we will also see no particular in the same interval. We suspect that this disparity is due to introduced for fooling DNN and LSAN (GPDNN) is almost classification. On the Spam dataset, we can craft adversarial examples for each combination of attacks, respectively. X denotes models excluded from evaluation.

| Dataset     | MNIST38 | MNIST91 | Hidost | Spam |
|-------------|---------|---------|--------|------|
| Attack      | JBM     | CW      | JBM    | CW   | JBM   | CW   |
| GPC         | 69      | 0       | 77     | 28.8 | -     | -    |
| GPDNN       | 0       | 0       | 0      | 1.1  | 0     | 0    |
| linDNN      | X       | X       | 49     | 0    | X     | 6    |
| DNN         | 0       | 0       | 0      | 0.4  | 0     | 0    |

ALGORITHMS CRAFTED BY JBM (JSMA, GPJM) AND CW RESPECTIVELY. X DENOTES FEATURES MODIFIED FOR SUCCESSFUL CARLINI WAGNER EXAMPLES. ALL NUMBERS OBSERVED IN A WHITE BOX SETTING.

| Dataset     | MNIST38 | MNIST91 | Hidost | Spam |
|-------------|---------|---------|--------|------|
| Attack      | JBM     | $L_\infty$ | JBM     | $L_\infty$ | JBM | $L_0$ | JBM | $L_2$ |
| GPC         | 2.76    | 1.47    | 2.02   | 6.85 | -     |
| GPDNN       | 1.01    | 0.04    | 4.01   | 0.05 | 1.01  | 1.14 | 3.62 | 1.2 |
| linDNN      | X       | X       | 4.26   | 0.08 | X     | 5.40 | 1.03 |
| DNN         | 1.13    | 1.39    | 1.85   | 1.09 | 0.68  | 0.01 | 3.79 | 1.19 |

On the Malware dataset, JSMA is not able to produce an adversarial example in around 1% of the test samples. On the same dataset, we also observe that LSAN (approximating GPLVM) need higher amount of perturbation to cause misclassification. On the Spam dataset, we can craft adversarial examples for every test point. The degree of perturbation introduced for fooling DNN and LSAN (GPDNN) is almost the same (3.79 for DNN, 3.62 for LSAN).

FGSM. On all datasets, we observe that the decrease in accuracy with increasing $\epsilon$ is much more pronounced on the DNN than on LSAN. This discrepancy is the most noticeable in the Spam dataset, where the accuracy drastically declines for DNN from 93.8% when $\epsilon = 0.001$ to 6.3% when $\epsilon = 0.4$. For LSAN (GPDNN), the accuracy falls from 90.1% to 22.7% in the same interval. We suspect that this disparity is due to gradient masking and should not be interpreted as LSAN being more robust. When investigating the performance of LSAN in the grey-box setting later on, we will also see no particular evidence for improved robustness.

CW/$L_\infty$. Due to the long runtime of these attacks, we crafted a set of 500 examples for each combination of attacks, classifiers and datasets. Across all datasets, we observe that the algorithms succeed in crafting adversarial examples for every test sample. We also measure the perturbations and depict them in Table III.

SVM-attack. Given the results in Table IV, we observe the strongest effect with examples crafted using the smallest $\epsilon = 0.001$ for all datasets. Also for all datasets, the accuracy in the worst case of this attack is roughly the same as a random guess. Compared to the other attacks, this attack is not very effective.

| Dataset     | $\epsilon$ | 0.001 | 0.01 | 0.1 | 0.2 | 0.3 | 0.4 |
|-------------|-------------|-------|------|-----|-----|-----|-----|
| M.38 GPC    | 95.0        | 94.0  | 92.6 | 94.4 | 95.4 | 50.4 |
| GPDNN       | 94.2        | 91.2  | 49.0 | 43.5 | 41.8 | 42  |
| lin SVM     | 48.3        | 48.5  | 49.1 | 49.1 | 49.1 | 49.1 |
| DNN         | 98.9        | 97.2  | 3.6  | 1.1  | 1.1  | 1.1 |
| M.91 GPC    | 99.8        | 99.8  | 90.6 | 51.0 | 51.0 | 51.0 |
| GPDNN       | 99.1        | 98.1  | 65.2 | 56.6 | 53.7 | 52.5 |
| lin SVM     | 50          | 50    | 52.2 | 52.9 | 52.9 | 52.9 |
| DNN         | 98.3        | 98.8  | 97.6 | 88.3 | 65.6 | 53.6 |
| linDNN      | 99.7        | 99.4  | 32.6 | 0.8  | 0.2  | 0.2 |
| Hidost GPC  | 99.2        | 99.2  | 98.8 | 94.6 | 94.6 | 94.6 |
| GPDNN       | 98.3        | 91.6  | 90.7 | 90.8 | 90.3 | 89.5 |
| lin SVM     | 91          | 92    | 95   | 95   | 95   | 95   |
| DNN         | 99.8        | 98.9  | 1.3  | 0.3  | 0.1  | 0.1  |
| Spam GPC    | 92.2        | 60.6  | 35.2 | 35.0 | 35.0 | 35.0 |
| GPDNN       | 90.1        | 60.2  | 24.9 | 23   | 23   | 22.7 |
| lin SVM     | 55.2        | 57.1  | 63.3 | 63.1 | 63.1 | 63.1 |
| linDNN      | 81.2        | 71.1  | 26.8 | 27.7 | 28.6 | 34.6 |
| DNN         | 93.8        | 63.1  | 7.7  | 6.3  | 6.3  | 6.3  |

GPJM and GPFGS. Similar to FGSM, a stronger $\epsilon$ leads to lower accuracy on the crafted adversarial examples. At the same time, there is often a point ($\epsilon = 0.2$ for Spam, $\epsilon = 0.2$ for MNIST91, $\epsilon = 0.1$ for MNIST38), where the accuracy remains almost constant. We then observe no further decrease in accuracy on crafted examples. For GPJM, we observe mixed results, where MNIST91 can be fooled completely. For other datasets, accuracy decreases to 28.8% (Spam) and 69% (MNIST38).

For both GP attacks it is necessary to invert the kernel matrix. This matrix very sparse for the Malware dataset, and the computation fails. We therefore use a pseudo-inverse to compute the attack. We still observe only a very moderate degree of success: only 27% of crafted examples are misclassified. We still answer RQ 1 positively—we are able to craft adversarial examples for GPC.

B. Transferability

We now investigate the transferability of the attacks described in the previous section. We start by investigating the capabilities of LSAN in approximating their counterparts. We then investigate GPC and transferability of the attack derived for, before turning to uncertainty estimates as a defense. For the interested reader, we provide the detailed results to all experiments in the Appendix, and use mainly meta statistics to ease understanding.

C. LSAN: Results of RQ 4 & RQ 6

We start by investigating whether LSAN enables attacks on models that use GPLVM latent space representations or kernels in general.

RQ 4. We want to verify whether LSAN are a good classifier for the problem setting their approximation target was trained in. We refer back to the results outlined in
When approximating GPLVM, we observe similar accuracies on MNIST91 (LSAN 99.2%, GPLVM 99.34%) and Spam (91.2% versus 91.8% originally). On Malware, LSAN is slightly better than the original by 0.3%. However, on MNIST38, it performs worse by more than three percent. In summary, all these approaches perform much better than random choice, and we thus answer RQ 4 positively. We will now investigate the efficiency when computing attacks on LSAN.

**RQ 5.** To evaluate this question, we compare the accuracy of the targeted algorithm on different kinds of adversarial examples. In particular, we compare these accuracies on the attacks computed on LSAN with all other grey box attacks (like DNN, linear SVM, GPC, and so on). More specifically, we subtract the accuracy on LSAN examples from the accuracy of all other related attacks. We then plot the distribution of these differences. If the majority of differences is positive, accuracy on adversarial examples computed on LSAN is higher (and thus the LSAN attack performs worse). We depict the results in Figure 3. Ideally, the accuracy of LSAN attacks should be lower, and consequently, the distributions in Figure 3 would be shifted towards the right-hand side.

We consider two settings: one for GPLVM and one for linear SVMs. We start with the results for GPLVM and SVM, approximated by GPDNN, shown in Figure 3a. For Malware and MNIST91 tasks, we observe that the differences are closely distributed around around zero from the left side. We also observe some outliers, e.g. for Malware we observe the GPJM to be much more effective than other techniques. The accuracy is roughly 61% as compared to over 90% for JSMA on GPDNN. On MNIST91, the linear SVM attack is more effective (reducing accuracy to 50% instead of more than 90%). For MNIST38, we observe that other attacks are more effective. For the Spam data, we observe large variance, indicating that no clear trend towards effectiveness for any particular technique exists. Yet, some positive outliers indicate that GPDNN are more effective than other attacks. This is mainly due to the bad performance of the linear SVM attack (accuracy > 90%) and GPJM (55% accuracy, other Jacobian attacks are < 20%) on this dataset, however.

We will now investigate the second setting, in particular, the approximation of a linear SVM by LSAN linDNN. We are left with two settings, MNIST91 and Spam, due to convergence. For MNIST91, data is distributed mostly towards the left-hand side, with strong variance, i.e. we conclude other attacks are more effective than LSAN. For Spam, we observe a more centralized distribution, with similar outliers due as observed for GPDNN.

Globally, we do observe however that all medians (gray) are close to 0, i.e. the attacks perform equally.

Given the previous results, we conclude that adversarial examples crafted on LSAN are as effective as other state-of-the-art algorithms in misleading the approximated technique.

We thus answer **RQ 5** negatively. A natural question at this point is whether LSAN differs at all from a normal DNN, i.e. whether the process of training a LSAN to approximate a substitute network produces a different result than training the DNN on the unprocessed data in the first place.

**RQ 6.** To evaluate this RQ, we rerun the evaluation on adversarial examples with a DNN that has the exact same architecture as the GPDNN. The DNN used in the grey box setting has the same architecture as the GPDNN. For the benign data where both are trained on, we already observed small differences in accuracy described in the previous section.

While the networks are trained on benign data, we are interested in their behavior on adversarial examples. We thus compare the difference between the accuracies for all types adversarial examples. Afterwards, we analyze the distribution of the deltas. The results for the four datasets and all three settings are depicted in Figure 4.

**a)** We first consider how much GPLVM+SVM and DNN differ. We depict these results in Figure 4a. We observe that for all datasets, the median is around 0. All distributions are centered around 0. Yet, the variances are large, and we observe differences in classification as large as 80 percentage points on the Spam data. For MNIST, these differences are in the worst
Fig. 4. Evaluating RQ 6. We compare the accuracies on adversarial examples of a GPDNN, DNN and GPLVM + SVM on adversarial examples. To obtain the distributions, we subtract accuracy from the second model from the first model for an attack α for all attacks. Hence on the right hand side, the first model has higher accuracy. On the left hand side, second model performs with higher accuracy. We also plot mean (black) and median (gray) values.

(a) DNN versus GPLVM+SVM

(b) GPLVM+SVM versus GPDNN

(c) DNN versus GPDNN

We compare the accuracies on adversarial examples of a GPDNN, DNN and GPLVM + SVM on adversarial examples. To obtain the distributions, we subtract accuracy from the second model from the first model for an attack α for all attacks. Hence on the right hand side, the first model has higher accuracy. On the left hand side, second model performs with higher accuracy. We also plot mean (black) and median (gray) values. Given the median is always around 0, we conclude that although in most cases similar, there can be large differences in the behavior.

b) We now compare GPLVM+SVM and GPDNN, e.g. the latent space classifier and its corresponding LSAN. The results are shown in Figure 4b. We observe that for all datasets except Spam, the distribution are dense around 0, e.g. in many cases where the output is very similar. Furthermore, the overall variance is small. There are some cases, however, where GPDNN performs with lower accuracy than GPLVM+SVM: On Spam data, the distribution is far away from 0 and centered at around 50 percentage points better. It is also rather uniform. We conclude that GPDNN approximates GPLVM+SVM rather closely on MNIST and the Malware dataset.

c) Finally, we investigate the differences between an LSAN and a DNN, visible in Figure 4c. We observe that for all datasets except Spam, the distribution are heavy around 0, e.g. there are many cases where the output is very similar. These findings are, however, not as pronounced as in the previous case. This is due to the higher variance caused by larger differences in the accuracy. For Spam, we observe GPDNN to consistently performs better than DNN. On the other dataset, there are some attacks where GPDNN is slightly better, however also some (MNIST) or many (Malware) settings where GPDNN performs much worse. In general, we observe differences of up to 50 (MNIST, Malware) or > 80 (Spam) percentage points.

Given the differences in classification for this range of attacks, we answer all parts of RQ 6 positively. There must be a difference in the inner representations to cause these differences in behavior. We also observe that LSAN is able to approximate the targeted technique. We now conclude our work on LSAN and move on to our attacks on GPC.

D. GPC: Results of RQ 2 and RQ 3

To investigate the transferability concerning GP, we consider two perspectives: First, RQ 2, we investigate whether adversarial examples crafted on GPC are misclassified by other algorithms. Second, RQ 3, we question whether adversarial examples crafted on other algorithms are misclassified by GPC.

We analyze both perspectives by analyzing the accuracies of both approaches on adversarial examples.

RQ 2. To evaluate this hypothesis, we compare the change in percentage of accuracy for all algorithms except GPC on benign test data and adversarial examples computed on GPC by GPJM and GPFGS

For all datasets, we conclude that the adversarial examples crafted for GPC are also misclassified by other algorithms. We further observe the average reduction is 20 percentage points for MNIST. For the Malware data, the average decrease is slightly higher, and around 27%. With > 40%, the average

Depending on the dataset, we consider linear SVM, RBF SVM, GPLVM+SVM, DNN, GPDNN, and linDNN. The last algorithm is only evaluated on MNIST91. RBF SVM is not evaluated on Spam.
We further plot the distribution of these differences. On the right hand side (of 0) accuracy decreases. We also plot mean (black) and median (gray). We observe that the decrease is largest in the Spam data. In some cases, we observe a small increases in accuracy, this is the case for Spam and MNIST data. These increases in accuracy are generally observed for GPFGS for small values of $\epsilon$, such as 0.001 and 0.01. Analogously to FGSM, a larger $\epsilon$ implies higher rate of misclassification.

**RQ 3** We will now investigate the opposite direction, e.g. transferability from adversarial examples crafted on other models on GPC. To answer this question, we subtract the accuracy on some adversarial examples (of all models, except GPC). In particular, we subtract the accuracy on adversarial examples from the accuracy on benign test data. We plot the distribution of these values. Hence, values $>0$ show a decrease of accuracy, $<0$ and increase of accuracy.

We observe that all datasets, attacks and algorithms we crafted on that they also mislead GPC. The average decrease is, compared to the previous findings, however smaller. Consider an average reduction of 30% versus $>40\%$ on the Malware data, and roughly 15% versus 20% on MNIST91. For MNIST38, the results are similar on average. For malware, we observe almost no reductions; however we in general observe (also for other models) that attacks have low transferability.

We further observe some cases where accuracy increases for adversarial examples on GPC. We observe this on Spam and MNIST38. On the Spam data, this is caused by the linear SVM attack. This attack is classified (by all models) correctly in all cases. On MNIST38, the cases where accuracy increases are FGSM examples with low $\epsilon$.

From the previous finding, we conclude that transferability holds in both directions for GPC. This refers to both adversarial examples crafted on other techniques exported to GPC and vice versa. We thus answer both **RQ 2** and **RQ 3** positively. Given this result, we confirm the threat of adversarial examples also for GP. On the other hand, there is a crucial difference between a DNN and any GP; uncertainty. We now turn to this property of GP and investigate whether it is able to alleviate the effect of adversarial examples.

**E. Uncertainty: Results of RQ 7 and RQ 8**

Intuitively, the variance of a prediction is low in areas where training data was observed. Accordingly, we might expect the variance to be larger in the vicinity of adversarial examples. Adversarial examples are likely to lie in regions away from training data: In high-dimensions the training data is likely contained on a low-dimensional manifold which is not orthogonal to the decision boundary. The creation of an adversarial example usually involves moving towards the boundary along a short path. However this process usually moves it away from the regions of training data, and thus towards areas with few data points, leading to increasing classification uncertainty. This hypothesis is consistent with the empirically validated assumption that adversarial examples stem from a distribution statistically separate from the training data. Hence, we have investigate the variance reported in the two GP approaches. In particular, we compare the mean variance across all adversarial examples and the mean variance across all benign data points. In the case of GPC, we also investigate the average absolute value of the latent function.

The intuition is that the behavior of this latent function is as well different in areas where no data was observed. The GPLVM, as an unsupervised ML method, does not use class labels in its initial training and its latent function is therefore interpreted differently. Here the value of the latent function is treated as the actual position in the transformed data space. We will start with GPC (RQ 7).

\footnote{For the RBF and many other, stationary kernels.}
The Carlini-Wagner (CW) algorithms. For the other attack metrics in adversarial examples crafted by
lightly, we see a decrease in variance for
larger the increase in variance of GPC. The variance also
that introduce global changes. The larger these changes, the
variance. The most noticeable changes are related to methods
are shown in Table VI. We can immediately observe that the
RQ 7. The averaged variance values for the tested cases
are shown in Table VI. We can immediately observe that the
different type of attacks have different degrees of impact on the
variance. The most noticeable changes are related to methods
that introduce global changes. The larger these changes, the
larger the increase in variance of GPC. The variance also
strongly increases for all Jacobian based methods. Interestingly,
we see a decrease in variance for $L_2$ and small increases for the
other attack metrics in adversarial examples crafted by the
Carlini-Wagner (CW) algorithms.

We also observe differences when monitoring the averaged
absolute latent function. The detailed results for this experiment
are shown in the Appendix.

We thus investigate a straight forward defense, which is
computing the 95% confidence interval of the data and
rejecting all data points that are outside this interval. Here, we
use both the latent variance and latent mean. We present our
results in Table V. We observe this simple step to be very
successful on the Spam data (except on the $L_x$ attacks). On
MNIST91 we observe mixed results. On MNIST38 and the
Malware data the approach does not work well. An additional
observation is that on the two datasets where the defense works
well, LSAN examples are in many cases harder to detect than
other adversarial examples.

To conclude, different methods for generating adversarial
to others. We investigated a straight forward defense, which is
computing the 95% confidence interval of the data and
rejecting all data points that are outside this interval. Here, we
use both the latent variance and latent mean. We present our
results in Table V. We observe this simple step to be very
successful on the Spam data (except on the $L_x$ attacks). On
MNIST91 we observe mixed results. On MNIST38 and the
Malware data the approach does not work well. An additional
observation is that on the two datasets where the defense works
well, LSAN examples are in many cases harder to detect than
other adversarial examples.

To conclude, different methods for generating adversarial
examples lead to different responses in the mean and the
variance of the GPC’s latent function. In a first step, we saw
that (depending on the data), we are able to exploit this as a

### Table V

| ORIGIN  | JBM | $L_0$ | $L_2$ | $L_{\infty}$ |
|---------|-----|------|------|-------------|
| MNIST91 | 6.382 |      |      |             |
| GPC     | 11.288 | 6.111 | 6.630 | 7.9870 | 11.9395 | 18.3882 | 27.2005 |
| GPDNN   | 10.0864 | 6.3756 | 6.3226 | 6.8299 | 9.9377 | 15.5973 | 23.6065 |
| lin SVM | 9.3149 | 6.3751 | 6.3152 | 6.2002 | 7.2703 | 9.5647 | 13.0987 |
| linDNN  | 11.0925 | 6.3755 | 6.3225 | 6.8148 | 9.8171 | 15.2891 | 23.0491 |
| DNN     | 103.1185 | 6.3407 | 6.5151 | 38.4048 | 131.4167 | 199.3153 | 215.8200 |
| GPDNN   | 132.7886 | 6.6645 | 6.5974 | 17.2383 | 57.2994 | 116.3130 | 162.1923 |
| lin SVM | 125.1362 | 6.6596 | 6.5997 | 19.3221 | 66.0221 | 131.8758 | 182.6009 |
| linDNN  | 129.9440 | 6.6452 | 6.4237 | 17.1022 | 60.0533 | 125.3152 | 179.3707 |

### Table VI

| ORIGIN  | JBM | $\epsilon = .001$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ |
|---------|-----|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| MNIST91 | 6.1561 | 6.3407 | 6.5151 | 38.4048 | 131.4167 | 199.3153 | 215.8200 |
| GPC     | 11.288 | 6.111 | 6.630 | 7.9870 | 11.9395 | 18.3882 | 27.2005 |
| GPDNN   | 10.0864 | 6.3756 | 6.3226 | 6.8299 | 9.9377 | 15.5973 | 23.6065 |
| lin SVM | 9.3149 | 6.3751 | 6.3152 | 6.2002 | 7.2703 | 9.5647 | 13.0987 |
| linDNN  | 11.0925 | 6.3755 | 6.3225 | 6.8148 | 9.8171 | 15.2891 | 23.0491 |
| DNN     | 103.1185 | 6.3407 | 6.5151 | 38.4048 | 131.4167 | 199.3153 | 215.8200 |
| GPDNN   | 132.7886 | 6.6645 | 6.5974 | 17.2383 | 57.2994 | 116.3130 | 162.1923 |
| lin SVM | 125.1362 | 6.6596 | 6.5997 | 19.3221 | 66.0221 | 131.8758 | 182.6009 |
| linDNN  | 129.9440 | 6.6452 | 6.4237 | 17.1022 | 60.0533 | 125.3152 | 179.3707 |
defense. It will be future work to investigate how to improve this by selecting other thresholds. Further we can influence the parameters learned for the covariance function. In our case, this function is similar to the RBF kernel, for which many security properties are known already [37].

Therefore, these measures mitigate some of the threat adversarial examples pose. We thus answer RQ 7 positively.

**RQ 8.** We show the variances of GPLVM for all kinds of adversarial examples in Table VII. For both MNIST tasks, we observe changes of +0.0001 or +0.0005 in the variance, if there are changes at all. For the Malware and Spam data, we do observe some changes: on the Spam data, the variance is an order of magnitude less. On the Malware data, variance shifts from 0.068 to 0.074 or 0.08.

As the degree of the differences is relatively insignificant, we suspect the variance provided in the predictions of GPLVM is not sufficiently informative about whether a data point is an adversarial example. We therefore cannot give a conclusive answer to RQ 8. Nevertheless, we might be able to modify GPLVM’s kernel to cause greater sensitivity to adversarial input. For example, shortening its lengthscale might force the variance to increase more quickly when moving away from training data locations. Investigating and determining the optimal hyper-parameters to alleviate the threat of adversarial examples for GPLVM is left as future work.

**VII. Related Work**

Transferability has been investigated in the context of adversarial examples has been brought up by Papernot et al. [31]. Rosza et al. [36] study transferability for different deep neural network architectures, whereas Liu et al. [24] investigate targeted transferability in particular. Finally Tramer et al. [44] explore transferability by examining the decision boundaries of different classifiers.

In contrast to all these works, we target explicitly security contexts with their binary setting. We further also focus on the usage of kernels and latent space and their effect on transferability.

Other, loosely related works included the work by Biggio et al. [4] and Papernot et al. [31]. Both introduce several attacks, among them attacks for SVM. In our work, we do not target SVM directly, but approximate them to apply common neural network based attacks.

To the best of our knowledge, there is only one approach to connect Gaussian processes to adversarial machine learning by Bradshaw et al. [6]. They do, however, only consider so-called Gaussian Hybrid networks, where the last layer of the neural network is replaced by a Gaussian process, and they evaluate the robustness of hybrid DNNs based only on FGSM and the attack by Carlini and Wagner. In contrast, our work targets Gaussian latent space models and Gaussian Process classification and studies transferability more extensively, using adversarial examples crafted by more algorithms and across several datasets.

**VIII. Conclusion**

In this paper, we explored the robustness of Gaussian Processes (GP), a machine learning model based on Bayesian Inference. We studied adversarial examples in this setting, coming to the following three main conclusions:
First, Gaussian Processes are as well vulnerable to adversarial examples. We constructed three novel attacks on GPs: Targeting GP classification in a supervised setting, we developed GPFGS and GPJM. Targeting GP Latent Variable models in an unsupervised setting, we proposed LSAN, a novel approach to approximating an arbitrary Latent Space representation.

Second, LSAN adversarial examples exhibit a less noticeable impact on uncertainty estimates while causing a similar amount of misclassifications. Further, LSAN approximate their target latent space models well in terms of behavior.

Third, Adversarial examples produced by state-of-the-art attacks noticeably affect uncertainty estimates in GP classification. We empirically investigated how adversarial examples affect the uncertainty estimates. In the case of GP classification, we observed significant changes to the latent function of GPC. We observe this simple defense to work reasonably well in two of four datasets. Finally, we implemented a straightforward defense based on the latent function of GPC. We observe this simple defense to work reasonably well in two of four datasets. Determining further configuration and effect of parameters learned by GPC is, however, left as future work.

ACKNOWLEDGMENT

This work was supported by the German Federal Ministry of Education and Research (BMBF) through funding for the Center for IT-Security, Privacy and Accountability (CISPA) (FKZ: 16KIS0753). This work has further been supported by the Engineering and Physical Research Council (EPSRC) Research Project EP/N014162/1.

REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016., pages 265–283, 2016.

[2] I. Andriotisopoulos, J. Koutsias, K. V. Chandrinos, G. Paliouras, and C. D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. arXiv preprint cs/0006013, 2000.

[3] M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. Machine Learning, 81(2):121–148, 2010.

[4] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part III, pages 387–402, 2013.

[5] B. Biggio, G. Fumera, F. Roli, and L. Didaci. Poisoning adaptive biometric systems. In Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, SSPR&SPR 2012, Hiroshima, Japan, November 7-9, 2012. Proceedings, pages 417–425, 2012.

[6] J. Bradshaw, A. G. d. G. Matthews, and Z. Ghahramani. Adversarial Examples, Uncertainty, and Transfer Testing Robustness in Gaussian Process Hybrid Deep Networks. ArXiv e-prints, July 2017.

[7] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. CoRR, abs/1608.04644, 2016.

[8] N. Carlini and D. A. Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. CoRR, abs/1705.07263, 2017.

[9] A. P. Engelbrecht. Using the Taylor expansion of multilayer feedforward neural networks. South African Computer Journal, 26:181–189, 2000.

[10] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. arXiv preprint arXiv:1202.2160, 2012.

[11] I. J. Goodfellow et al. Explaining and harnessing adversarial examples. In Proceedings of the 2015 International Conference on Learning Representations, 2015.

[12] I. J. Goodfellow, N. Papernot, and P. D. McDaniel. cleverhans v0.1: an adversarial machine learning library. CoRR, abs/1610.00768, 2016.

[13] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel. On the (Statistical) Detection of Adversarial Examples. ArXiv e-prints, Feb. 2017.

[14] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. CoRR, abs/1606.04435, 2016.

[15] Y. Han and B. I. P. Rubinstein. Adequacy of the Gradient-Descent Method for Classifier Evasion Attacks. ArXiv e-prints, Apr. 2017.

[16] W. Hu and Y. Tan. Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN. ArXiv e-prints, Feb. 2017.

[17] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In International Conference on Computer Aided Verification, pages 3–29, Springer, 2017.

[18] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. [Online; accessed 2017-08-21].

[19] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. CoRR, abs/1702.01135, 2017.

[20] R. Laishram and V. V. Phoha. Curie: A method for protecting SVM classifier from poisoning attack. CoRR, abs/1606.01584, 2016.

[21] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In Advances in neural information processing systems, pages 329–336, 2004.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In Proceedings of the IEEE, pages 2278–2324, 1998.

[23] M. Lichman. UCI machine learning repository, 2013.

[24] Y. Liu, X. Chen, C. Liu, and D. Song. Delving into transferable adversarial examples and black-box attacks. CoRR, abs/1611.02770, 2016.

[25] S. M. Loos, G. Irving, C. Szegedy, and C. Kaliszyn. Deep network guided proof search. CoRR, abs/1701.06972, 2017.

[26] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In CEAS 2005 - Second Conference on Email and Anti-Spam, July 21-22, 2005, Stanford University, California, USA, 2005.

[27] D. Maiorca, I. Corona, and G. Giacinto. Looking at the bag is not enough to find the bomb: an evasion of structural methods for malicious PDF files detection. In 8th ACM Symposium on Information, Computer and Communications Security, ASIA CCS ’13, Hangzhou, China - May 08 - 10, 2013, pages 119–130, 2013.

[28] M. McCoyd and D. Wagner. Spoofing 2D Face Detection: Machines See People Who Aren’t There. ArXiv e-prints, Aug. 2016.

[29] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method for fool deep neural networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[30] N. Narodytska and S. P. Kasiviswanathan. Simple black-box adversarial perturbations for deep networks. CoRR, abs/1612.06299, 2016.

[31] N. Papernot, P. McDaniel, and I. J. Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial perturbations for deep networks. CoRR, abs/1605.07277, 2016.

[32] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The Limitations of Deep Learning in Adversarial Settings. In Proceedings of the 1st IEEE European Symposium in Security and Privacy (EuroS&P), 2016.

[33] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman. Towards the science of security and privacy in machine learning. CoRR, abs/1611.03814, 2016.

[34] C. E. Rasmussen and C. K. I. Williams. Gaussian processes for machine learning. Adaptive computation and machine learning. MIT Press, 2006.
[35] K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4):639–668, 2011.

[36] A. Rozsa, M. Günther, and T. E. Boult. Are Accuracy and Robustness Correlated? *ArXiv e-prints*, Oct. 2016.

[37] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli. Secure kernel machines against evasion attacks. In *AISec@CCS*, pages 59–69. ACM, 2016.

[38] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *10th International Conference on Malicious and Unwanted Software, MALWARE 2015, Fajardo, PR, USA, October 20-22, 2015*, pages 11–20, 2015.

[39] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE symposium on security and privacy*, pages 305–316. IEEE, 2010.

[40] N. Šrndić and P. Laskov. Practical evasion of a learning-based classifier: A case study. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 197–211, 2014.

[41] N. Šrndić and P. Laskov. Hidost: a static machine-learning-based detector of malicious files. *EURASIP Journal on Information Security*, 2016(1):22, Sep 2016.

[42] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *Proceedings of the 2014 International Conference on Learning Representations*. Computational and Biological Learning Society, 2014.

[43] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[44] F. Trančer, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel. The Space of Transferable Adversarial Examples. *ArXiv e-prints*, Apr. 2017.

[45] F. Trančer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 601–618, 2016.

[46] P. Vidnerová and R. Neruda. Vulnerability of machine learning models to adversarial examples. In *Proceedings of the 16th ITAT Conference Information Technologies - Applications and Theory, Tatranské Matliare, Slovakia, September 15-19, 2016.*, pages 187–194, 2016.

[47] W. Xu, Y. Qi, and D. Evans. Automatically evading classifiers. In *Proceedings of the 2016 Network and Distributed Systems Symposium*, 2016.
Gray box evaluation on MNIST38 for all described attacks. Percentage indicated how many adversarial examples are correctly classified by the model in target, when crafted on origin using the corresponding attack. JBM denotes Jacobian-based methods, such as JSMA on DNN or GPJM for GPC. — denotes that an attack was not implemented for a given algorithm. Methods with insufficient accuracy (Table II) are excluded from evaluation.

| origin | target | JBM | $\epsilon = .001$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ | $l_2$ | $l_1$ |
|--------|--------|-----|------------------|------------------|------------------|------------------|------------------|------------------|-------|-------|
| GPC    | 72.9   | 95.6| 95.8            | 95.6            | 95.8            | 84.2            | 50.4            | -                | -     | -     |
| GPDNN  | 65.2   | 93.2| 93.6            | 88.8            | 80.8            | 71.8            | 68.6            | -                | -     | -     |
| GPLVM  | 75.5   | 97.2| 97.4            | 97.2            | 94.8            | 93.0            | 89.6            | -                | -     | -     |
| GPC    | 76.8   | 97.4| 96.4            | 78.0            | 65.2            | 58.6            | 54.8            | -                | -     | -     |
| lin SVM| 77.4   | 98.0| 98.0            | 94.4            | 87.8            | 82.2            | 78.6            | -                | -     | -     |
| DNN    | 81.9   | 99.2| 99.2            | 95.2            | 83.2            | 73.6            | 65.0            | -                | -     | -     |
| GPC    | 93.8   | 94.7| 94.8            | 94.7            | 91.3            | 66.4            | 49.0            | 53.2             | 94.2  | -     |
| GPDNN  | 94.2   | 93.8| 93.9            | 90.8            | 81.4            | 68.2            | 57.9            | 52.6             | 94.6  | -     |
| GPLVM  | 97.7   | 97.6| 97.7            | 97.7            | 97.3            | 95.4            | 90.3            | 53.4             | 98.4  | -     |
| GPDNN  | 92.7   | 96.9| 96.7            | 86.0            | 75.4            | 74.1            | 73.9            | 52.8             | 97.2  | -     |
| lin SVM| 96.9   | 97.4| 97.3            | 96.8            | 93.8            | 88.2            | 80.0            | 52.6             | 97.4  | -     |
| DNN    | 98.4   | 98.7| 98.7            | 97.8            | 94.2            | 85.7            | 73.2            | 52.8             | 98.6  | -     |
| GPC    | —      | 47.7| 47.6            | 47.8            | 48.4            | 48.2            | 49.0            | -                | -     | -     |
| GPDNN  | —      | 48.5| 48.5            | 49.3            | 50.9            | 50.8            | 48.1            | -                | -     | -     |
| GPLVM  | —      | 48.2| 48.1            | 48.2            | 48.4            | 48.6            | 48.6            | -                | -     | -     |
| lin SVM| —      | 48.3| 48.5            | 49.1            | 49.1            | 49.1            | 49.1            | -                | -     | -     |
| RBF SVM| —      | 48.2| 48.3            | 48.5            | 49.1            | 49.3            | 49.2            | -                | -     | -     |
| DNN    | —      | 48.3| 48.3            | 48.8            | 48.6            | 49.3            | 49.1            | -                | -     | -     |
| GPC    | 90.8   | 94.7| 94.6            | 93.1            | 89.1            | 64.7            | 49.1            | 54.4             | 95.8  | -     |
| GPDNN  | 91.0   | 93.8| 93.6            | 89.0            | 77.5            | 61.8            | 48.5            | 66.6             | 93.2  | -     |
| GPLVM  | 97.3   | 97.6| 97.6            | 97.0            | 94.6            | 87.2            | 71.6            | 43.2             | 98.2  | -     |
| DNN    | 84.9   | 96.9| 95.9            | 59.0            | 47.5            | 44.9            | 41.3            | 53.2             | 96.6  | -     |
| RBF SVM| 95.8   | 97.4| 97.1            | 91.1            | 68.0            | 37.8            | 17.8            | 59.4             | 97.2  | -     |
| DNN    | 97.4   | 98.7| 98.6            | 91.6            | 51.8            | 22.3            | 8.6             | 47.4             | 98.8  | -     |
| origin | target       | JBM   | FGSM / linSVM / GPFGS | CW |
|--------|--------------|-------|-----------------------|----|
|        |              |       | $\epsilon = .001$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ |
| MNIST91 |             |       | $l_2$ | $l_4$ |
| GPC    |              | 97.4  | 100.0 | 100.0 | 97.0  | 51.0  | 51.0  | 51.0  | – | – |
| GPDNN  |              | 97.4  | 100.0 | 100.0 | 95.2  | 55.4  | 52.2  | 51.8  | – | – |
| GPVM   |              | 98.2  | 99.4  | 99.3  | 99.6  | 99.6  | 99.0  | 94.6  | – | – |
| GPC    | lin SVM      | 84.2  | 99.8  | 99.8  | 61.8  | 51.0  | 51.0  | 51.0  | – | – |
| RBF SVM|              | 98.8  | 100.0 | 100.0 | 93.0  | 51.0  | 51.0  | 51.0  | – | – |
| linDNN |              | 89.6  | 99.0  | 98.6  | 87.0  | 51.0  | 51.0  | 51.0  | – | – |
| DNN    |              | 92.8  | 100.0 | 100.0 | 83.4  | 51.0  | 51.0  | 51.0  | – | – |
| GPC    |              | 99.5  | 99.6  | 99.6  | 99.6  | 99.5  | 98.0  | 84.0  | 56.6 | 99.4 |
| GPDNN  |              | 98.8  | 99.3  | 99.2  | 97.2  | 69.0  | 50.2  | 47.4  | 45.4 | 99.4 |
| GPVM   |              | 99.3  | 99.3  | 99.3  | 99.3  | 99.3  | 99.1  | 98.8  | 54.6 | 99.6 |
| GPDNN  | lin SVM      | 99.2  | 99.5  | 99.5  | 99.2  | 95.8  | 85.3  | 68.0  | 48.0 | 99.6 |
| RBF SVM|              | 99.6  | 99.6  | 99.6  | 99.6  | 99.4  | 96.2  | 99.8  | 48.8 | 99.6 |
| linDNN |              | 98.2  | 98.6  | 98.4  | 97.6  | 91.5  | 69.8  | 53.4  | 48.4 | 98.2 |
| DNN    |              | 99.6  | 99.7  | 99.7  | 99.7  | 97.7  | 82.8  | 65.0  | 60.6 | 99.8 |
| GPC    |              | –     | 50.1  | 50.1  | 50.3  | 49.9  | 49.9  | 50.4  | – | – |
| GPDNN  |              | –     | 50.1  | 50.1  | 50.0  | 48.6  | 47.4  | 47.1  | – | – |
| GPVM   |              | –     | 50.0  | 50.0  | 50.0  | 50.0  | 50.0  | 50.0  | – | – |
| lin SVM|              | –     | 50.0  | 50.0  | 52.4  | 52.9  | 52.9  | 52.9  | – | – |
| RBF SVM|              | –     | 50.2  | 50.1  | 50.0  | 50.0  | 50.5  | 51.3  | – | – |
| linDNN |              | –     | 50.0  | 50.0  | 50.1  | 50.0  | 49.9  | 49.9  | – | – |
| DNN    |              | –     | 50.1  | 50.1  | 50.2  | 50.6  | 50.5  | 51.4  | – | – |
| GPC    |              | 99.6  | 99.6  | 99.5  | 99.3  | 98.0  | 93.0  | 77.2  | 99.8 | 99.8 |
| GPDNN  |              | 94.8  | 99.3  | 99.2  | 98.9  | 97.9  | 96.2  | 90.1  | 47.6 | 99.0 |
| GPVM   |              | 98.2  | 99.4  | 99.3  | 99.6  | 98.6  | 98.1  | 95.8  | 54.0 | 99.8 |
| linDNN |              | 82.4  | 99.5  | 99.5  | 99.3  | 98.5  | 96.5  | 91.4  | 94.8 | 99.8 |
| RBF SVM|              | 99.5  | 99.6  | 99.5  | 99.5  | 98.1  | 94.1  | 99.8  | 98.2 | 99.6 |
| linDNN |              | 99.4  | 98.6  | 98.6  | 97.2  | 88.6  | 67.8  | 53.8  | 100.0 | 99.8 |
| DNN    |              | 96.2  | 99.7  | 99.7  | 99.4  | 98.1  | 96.1  | 91.9  | 60.8 | 99.8 |
| GPC    |              | 99.6  | 99.6  | 99.6  | 99.1  | 93.2  | 54.3  | 40.7  | 99.8 | 100.0 |
| GPDNN  |              | 98.6  | 99.3  | 99.2  | 96.5  | 70.8  | 52.5  | 50.5  | 47.4 | 99.4 |
| GPVM   |              | 99.3  | 99.3  | 99.3  | 99.2  | 99.0  | 98.6  | 96.5  | 53.4 | 99.4 |
| DNN    |              | 97.1  | 99.5  | 99.4  | 90.7  | 40.4  | 22.6  | 9.4  | 95.8 | 99.6 |
| RBF SVM|              | 99.6  | 99.6  | 99.5  | 99.3  | 83.6  | 45.1  | 40.2  | 100.0 | 99.8 |
| linDNN |              | 96.0  | 98.5  | 98.4  | 94.3  | 63.0  | 46.0  | 45.4  | 93.0 | 98.8 |
| DNN    |              | 99.7  | 99.7  | 99.8  | 98.8  | 63.3  | 44.5  | 41.1  | 61.0 | 100.0 |

**TABLE IX**

Gray box evaluation on MNIST91 for all described attacks. Percentage indicated how many adversarial examples are correctly classified by the model in target, when crafted on origin using the corresponding attack. JBM denotes Jacobian based methods, such as JSMA on DNN or GPJM for GPC. – denotes that an attack was not implemented for a given algorithm. Methods with insufficient accuracy (Table IX) are excluded from evaluation.
### Table X

Gray box evaluation on the Malware dataset for all described attacks. Percentage indicated how many adversarial examples are correctly classified by the model in target, when crafted on origin using the corresponding attack. JBM denotes Jacobian based methods, such as JSMA on DNN or GPJM for GPC. -- denotes that an attack was not implemented for a given algorithm. Methods with insufficient accuracy (Table I) are excluded from evaluation. The GPC attacks on this dataset were computed using a pseudo-inverse due to sparsity of the data.

| Malware | origin | target | JBM | FGSM / linSVM / GPFGS | CW |
|---------|--------|--------|-----|------------------------|----|
|         |        |        | $\epsilon = .001$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ |
|         |        |        | $l_0$ | $l_1$ | $l_2$ |
| DNN     | GPC    | 98.5   | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 94.8  | 98.2 |
|         | GPDNN  | 87.4   | 86.8  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 78.6  |
| GPLVM   | 87.1   | 86.6  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| lin SVM | 86.8   | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  |
| RBF SVM | 87.1   | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| DNN     | GPC    | 98.5   | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 94.8  | 98.2 |
|         | GPDNN  | 87.4   | 86.8  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 78.6  |
| GPLVM   | 87.1   | 86.6  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| lin SVM | 86.8   | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  |
| RBF SVM | 87.1   | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |

### Table XI

Gray box evaluation on the Spam dataset for all described attacks. Percentage indicated how many adversarial examples are correctly classified by the model in target, when crafted on origin using the corresponding attack. JBM denotes Jacobian based methods, such as JSMA on DNN or GPJM for GPC. -- denotes that an attack was not implemented for a given algorithm. Methods with insufficient accuracy (Table I) are excluded from evaluation.

| Spam   | origin | target | JBM | FGSM / linSVM / GPFGS | CW |
|--------|--------|--------|-----|------------------------|----|
|        |        |        | $\epsilon = .001$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ |
|        |        |        | $l_0$ | $l_1$ | $l_2$ |
| DNN    | GPC    | 98.5   | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 94.8  | 98.2 |
|         | GPDNN  | 87.4   | 86.8  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 78.6  |
| GPLVM  | 87.1   | 86.6  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| lin SVM | 86.8   | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  |
| RBF SVM | 87.1   | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| DNN    | GPC    | 98.5   | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 98.5  | 94.8  | 98.2 |
|         | GPDNN  | 87.4   | 86.8  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 87.4  | 78.6  |
| GPLVM  | 87.1   | 86.6  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
| lin SVM | 86.8   | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  | 86.8  |
| RBF SVM | 87.1   | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  | 87.1  |
### TABLE XII
**Average absolute latent function in GPC for benign data (bold) and adversarial examples crafted by algorithm ORIGIN**

| ORIGIN   | JBM | $\epsilon = .01$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ | $L_0$ | $L_2$ | $L_\infty$ |
|----------|-----|------------------|------------------|------------------|------------------|------------------|------------------|-------|-------|-------------|
| MNIST38  | 1.2e-08 | 2.5e-08 | 2.2e-08 | 1.7e-09 | 1.3e-11 | 2.8e-15 | 7.4e-21 | – | – | – |
| GPC      | 5.9e-10 | 1.2e-08 | – | 6.9e-09 | 6.9e-09 | 6.9e-09 | – | 4.1e-10 | 6.3e-09 | – |
| GPNN     | 5.0e-10 | 3.0e-05 | 6.9e-06 | 1.1e-07 | 1.4e-10 | 2.4e-14 | – | – | – |
| lin SVM  | 2.1e-10 | 2.2e-08 | 2.2e-08 | 2.9e-06 | 3.6e-09 | 4.1e-19 | 6.8e-33 | – | – | – |
| DNN      | 2.1e-10 | 1.2e-08 | 2.0e-08 | 1.4e-09 | 3.5e-11 | 3.5e-11 | 3.5e-11 | 0.3504 | – | – |
| MALW     | 0.3107 | 0.3310 | 0.3058 | 0.0009 | 2.5e-11 | 8.1e-24 | 2.9e-21 | – | – | – |
| GPC      | 0.0115 | 0.2902 | 0.2856 | 0.0060 | 0.0425 | 0.0425 | 0.0425 | 0.2689 | – | – |
| GPNN     | 0.2410 | 3.4084 | 3.4445 | 3.1139 | 2.8010 | 2.7574 | 1.9898 | – | – | – |
| DNN      | 1.7944 | 2.4834 | 3.4399 | 3.0198 | 2.5746 | 2.1854 | 1.9898 | – | – | – |
| DNN      | 2.7306 | 3.4766 | 3.3723 | 2.3512 | 1.3243 | 1.1260 | 1.5329 | – | – | – |
| SPAM     | 3.4721 | 0.3305 | 0.3095 | 0.0015 | 2.5e-11 | 8.1e-24 | 2.9e-21 | – | – | – |
| GPC      | 3.4721 | 0.3305 | 0.3095 | 0.0015 | 2.5e-11 | 8.1e-24 | 2.9e-21 | – | – | – |
| GPNN     | 0.3405 | 2.9515 | 2.9515 | 1.8452 | 1.4223 | 1.4223 | 1.4223 | – | – | – |
| DNN      | 1.9473 | 2.7505 | 7.4281 | 10.9106 | 8.6904 | 4.7270 | 3.7479 | 1.1334 | 3.8841 | – |
| DNN      | 12.3196 | 3.3805 | 2.2771 | 12.1908 | 17.6029 | 13.9747 | 7.5925 | 3.8349 | 0.6372 | 3.5961 |

### TABLE XIII
**Standard deviation of absolute latent function for benign data (bold) and adversarial examples in GPC. Adversarial examples crafted by algorithm ORIGIN.**

| ORIGIN   | JBM | $\epsilon = .01$ | $\epsilon = .01$ | $\epsilon = .1$ | $\epsilon = .2$ | $\epsilon = .3$ | $\epsilon = .4$ | $L_0$ | $L_2$ | $L_\infty$ |
|----------|-----|------------------|------------------|------------------|------------------|------------------|------------------|-------|-------|-------------|
| text8MNIST38 | 3.1e-27 | 4.4e-26 | 2.6e-26 | 1.5e-32 | 4.8e-42 | 1.2e-55 | 1.1e-77 | – | – | – |
| GPC      | 2.6e-33 | 3.2e-27 | 3.2e-27 | 3.0e-27 | 3.0e-27 | 3.0e-27 | 3.0e-27 | – | 2.5e-36 | 3.3e-35 |
| GPNN     | 1.3e-33 | 3.0e-16 | 9.0e-19 | 5.9e-26 | 4.4e-37 | 2.6e-51 | – | 0.4280 | 3.5067 | – |
| DNN      | 2.3e-33 | 3.2e-27 | 3.0e-27 | 1.5e-30 | 7.1e-41 | 4.8e-57 | 8.1e-79 | – | 5.7e-35 | 9.8e-31 |
| MALW     | 0.055 | 0.0585 | 0.0439 | 3.8e-12 | 3.7e-42 | 5.3e-92 | 1.2e-161 | – | – | – |
| GPC      | 9.7e-10 | 0.0403 | 0.0371 | 0.0007 | 0.0007 | 0.0007 | 0.0007 | 0.0408 | – | – |
| GPNN     | 0.0005 | 0.0456 | 0.0456 | 5.9e-10 | 6.9e-34 | 3.4e-73 | 1.8e-125 | – | – | – |
| DNN      | 1.5e-06 | 0.0403 | 0.0366 | 2.6e-07 | 5.4e-23 | 1.2e-38 | 1.2e-38 | 0.0724 | – | – |
| MNIST91  | 0.88 | 0.8576 | 1.2861 | 68.3405 | 159.9887 | 137.8570 | 112.8514 | – | – | – |
| GPC      | 0.0324 | 0.8576 | 1.2861 | 68.3405 | 159.9887 | 137.8570 | 112.8514 | – | – | – |
| GPNN     | 0.9988 | 0.8809 | 0.9888 | 0.8388 | 1.3636 | 2.0621 | 2.8802 | – | 0.0006 | 0.8321 |
| DNN      | 0.0925 | 0.8721 | 0.8073 | 0.5994 | 1.1210 | 2.7048 | 4.1999 | – | 0.0031 | 0.9014 |
| DNN      | 0.8581 | 0.8817 | 0.8970 | 1.0513 | 1.1125 | 0.2479 | 0.6343 | – | 0.0014 | 0.8501 |
| SPAM     | 63.9184 | 76.8750 | 92.2008 | 318.4397 | 39.9429 | 0.6991 | 0.0012 | – | – | – |
| GPC      | 40.9779 | 66.3834 | 347.6565 | 945.0137 | 284.5576 | 44.7802 | 62.0666 | 0.0469 | 64.3927 | – |
| GPNN     | 53.2261 | 67.7945 | 66.3834 | 347.6565 | 945.0137 | 284.5576 | 44.7802 | 62.0666 | 0.0469 | 64.3927 | – |
| DNN      | 6.4378 | 64.9965 | 49.1872 | 178.5613 | 530.6283 | 347.7331 | 70.0789 | 83.4065 | 1.9466 | 71.6031 | – |
| DNN      | 113.8493 | 65.5118 | 36.6047 | 99.7421 | 111.9318 | 67.0232 | 17.7011 | 68.9255 | 0.1808 | 57.8335 | – |