Defending Evasion Attacks via Adversarially Adaptive Training

Minh-Hao Van, Wei Du, Xintao Wu
University of Arkansas
Fayetteville, AR, USA
{haovan, wd005, xintaowu}@uark.edu

Feng Chen
University of Texas at Dallas
Dallas, TX, USA
feng.chen@utdallas.edu

Aidong Lu
University of North Carolina at Charlotte
Charlotte, NC, USA
alu1@uncc.edu

***Abstract*—Adversarial machine learning has been extensively studied from perspectives of attack settings and defense strategies. However, existing adversarial training models fail to be adaptive and robust against new attacks during test time. In this paper, we propose a novel adversarially adaptive defense (AAD) framework based on adaptive training such that the trained prediction and detection models adapt at test time to new attacks. Our AAD structures the training data into groups and each group represents one attack scenario. Different from empirical risk minimization that trains a single robust model or learns an invariant feature space, our AAD learns a context vector from features of each batch during training and incorporates the learned context vector into both prediction and detection models. Thus, AAD can adapt at test time to new adversarial attacks. We formulate our problem by optimizing a joint loss from prediction, detection, and regularization via a multi-task learning framework. We conduct comprehensive empirical evaluations with popular adversarial attacks and defense strategies on two real-world datasets under different attack settings. Empirical results show that AAD achieves both high prediction and detection accuracy and significantly outperforms baselines.**

***Index Terms*—test time attack, adaptive training, adversarial machine learning**

I. Introduction

Adversarial machine learning focuses on vulnerabilities in machine learning models and has been extensively studied from perspectives of attack settings and defense strategies (see surveys [1]–[3]). Adversarial training is one of the widely used defense approaches and aims to train a robust model by injecting adversarial examples into training data. However, as pointed out in [4], it is essential to include adversarial examples produced by as many as known attacks in adversarial training such that the deployed model could be potentially robust at test time. There are two inherent challenges. First, it is computationally expensive to find adversarial inputs by the most known attacking techniques. Second, there are always new or unknown attacks at test time. Most of the current defense strategies are based on empirical risk minimization (ERM) and assume the data distribution at test time matches the training distribution. These methods are not adaptive to new adversarial attacks as defense methods that are built to block one kind of attack leave another vulnerability open to other attacks. It is imperative to develop a defense model that is robust and adaptive to new test time attacks.

978-1-6654-8045-1/22/$31.00 ©2022 IEEE

In this paper, we focus on the classification task in the presence of test time attacks. In particular, we focus on evasion attacks that aim to degrade the performance of deployed models, causing an increase of false positives and false negatives. We leverage adversarial training (i.e., including adversarial examples generated from known attacks) to develop a novel adaptive defense strategy that is robust under both known and unknown test time attacks. Our goal is to strengthen the model robustness and improve predictive performance at the test stage. Inspired by adaptive risk minimization (ARM) [5], we develop the adversarially adaptive defense (AAD) framework such that the trained prediction and detection models adapt at test time to new attacks. We structure the training data into groups and each group represents one attack scenario. Different from ERM that trains a single robust model or learns an invariant feature space, our AAD learns a context vector from features of each batch during the training and incorporates the learned context vector into both prediction and detection tasks. Note that the context vector captures group contextual information such as dynamics in a batch. During the test time, AAD first learns the context vector of a coming batch and then combines it with data examples as input to both prediction and detection models. Thus, AAD can adapt at test time to new adversarial attacks.

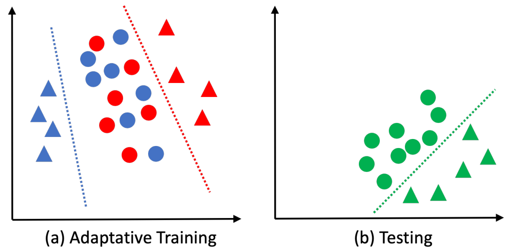


Fig. 1: Adaptive training illustration. There are two groups in the training set and one group in the testing. Each group represents one attack scenario and is indicated by a different color. Clean examples are denoted as a circle and adversarial examples as a triangle. Adversarial examples in one group are produced by one particular attack.

Figure 1 shows a toy example to illustrate our AAD. During the training, we have two groups and each group consists of clean examples and adversarial examples generated by one particular attack. We consider each group to represent one attack scenario. We have a different attack scenario during the test time. In each attack scenario, the input-output relationship varies. Hence, techniques such as robust optimization and learning an invariant feature space, e.g., [6], which try to derive one consistent input-output relationship across all groups, are not applicable. On the contrary, our AAD can achieve adaption to groups in training and the new group at test time.

Our contributions are summarized as follows. First, we propose AAD, a novel adversarial defense framework that is robust and adaptive to new test time attacks. Second, we develop a method to compute the joint loss from prediction, detection, and regularization via a multi-task learning framework. Third, we conduct comprehensive empirical evaluations with 18 popular adversarial attacks, eight defense baselines, and two real-world datasets with different settings of attack scenarios. Empirical results show that AAD achieves both high prediction and detection accuracy and outperforms all defense baselines with a p-value less than 0.01.

II. Related Work

A. Adaptive Learning

Recent research on robust machine learning has achieved significant progress. One of the emerging approaches is test time adaptation to tackle distribution shift. Traditional machine learning algorithms typically assume that the training and test data are generated from the same distribution. In real applications, this assumption is highly unrealistic due to distribution shift over time or unseen data. In [5], Zhang et al. proposed a test time adaptation model to group shift called adaptive risk minimization for the setting where the training data are structured into groups and test time data represents new groups or new group distributions. ARM aims to learn models that adapt at test time to shift using unlabeled test points. Specifically, they introduced a meta-learning algorithm in which models are optimized for post adaption performance on training batches sampled from different groups. The sampling process simulates group shifts that may occur at test time. To deal with the similar test time adaption problem, Wang et al. introduced a new method using entropy minimization called *tent* [7]. The objective is to minimize the entropy of the predictions from model. Different from [5], [7], our AAD considers each attack scenario as a group, learns the group's context vector, and then incorporates it in prediction and detection tasks.

B. Adversarial Machine Learning

1) Adversarial Attacks: The bulk of recent research on adversarial machine learning has focused on test time attacks where the attacker perturbs the test data to obtain a desired classification. In evasion attack, the adversary tries to evade the system by adjusting malicious examples and forcing the model to produce incorrect outputs during test phase. Test time attacks can be classified into white-box or black-box attacks. Train time attacks leave the test data unchanged, and instead perturb the training data to affect the learned model. Data poisoning attacks are among the most common train time attack methods in adversarial learning. Approaches include optimization-based methods [8] (e.g., influence, KKT, and min-max attack), poisoning Generative Adversarial Net (pGAN) model [9], and class-oriented poisoning attacks against neural network models [10]. Attacks can be either targeted or non-targeted attacks where targeted attacks maximize the probability of targeted adversarial class and non-targeted attacks do not assign a specific class to the classifier's output. Attacks can also be white-box or black-box where white-box attacks have complete access to the model, including its structure and weights, and black-box attacks can only access the probability or the label of a given input. Black-box attacks can be further split into score-based and decision-based. A score-based attack assumes access to the output layer and a decision-based attack assumes access to only the predicted label. In the supplementary file, we present details of 18 attacks used in our implementation.

2) Adversarial Defense: Defense methods have three main categories: adversarial training (or robust optimization), gradient masking, and adversarial example detection. Adversarial training [11]–[13] trains a robust model on a training set augmented with adversarially perturbed data. Its goal is to improve the classification accuracy of the target model on adversarial examples while maintaining accuracy on normal examples. Gradient masking, e.g., defensive distillation [14], is a defense strategy where a defender deliberately hides the gradient information of the model to confuse the adversary. It applies regularizers or smooth labels to make the model output less sensitive to the perturbation on input. Adversarial example detection methods [15], [16] exploit adversarial examples to train a detector. These methods typically require additional information, e.g., a labeled set of outliers or a clean set, and apply supervised classification to separate outliers from normal examples. Moreover, these methods do not generalize well across different attack parameters and attack generation processes. In Section IV-B, we present details of eight defense methods used as our baselines. Our AAD is among few methods, e.g., MagNet [15] and Feature Squeezing [16], that can achieve both prediction and detection, and is the only one that exploits adaptive training to achieve robustness against new test time attacks. Very recently, transductive learning based defense [17] was proposed to achieve adversarial robustness. However, the method needs to dynamically retrain the model based on test time input and requires solving expensive bilevel attack objectives. Our AAD remains unchanged after deployment and learns context vectors to adapt to new attacks.

III. Adaptive Training Framework

A. Problem Formulation

In this section, we formulate and present our adversarially adaptive defense framework. Let $\mathcal{D}$ be the training dataset consisting of two types of data, clean data $\mathcal{D}_c$ and attack data $\mathcal{D}_a$ generated from a set of known attacks. The goal of introducing $\mathcal{D}_a$ is to increase robustness of the trained model

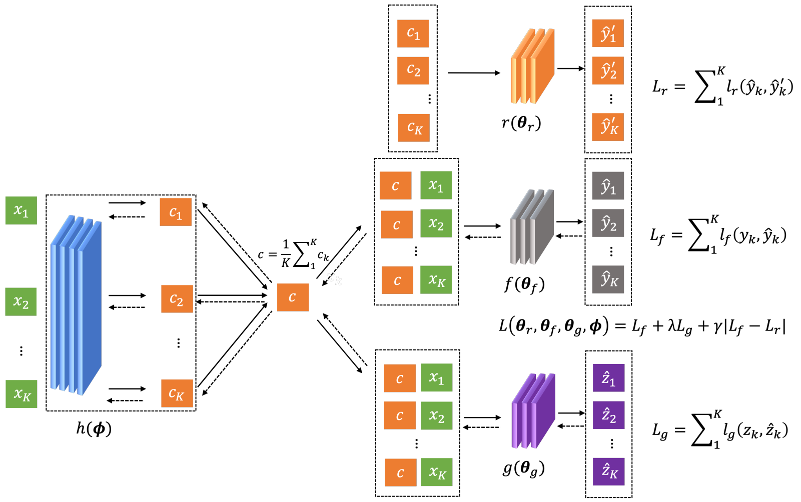

c_1
c_2
c_K
$r(\boldsymbol{\theta}_r)$
$\hat{y}'_1$
$\hat{y}'_2$
$\hat{y}'_K$
$L_r = \sum_1^K l_r(\hat{y}_k, \hat{y}'_k)$
x_1
x_2
x_K
$h(\boldsymbol{\phi})$
$c = \frac{1}{K}\sum_1^K c_k$
c
$f(\boldsymbol{\theta}_f)$
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_K$
$L_f = \sum_1^K l_f(y_k, \hat{y}_k)$
$L(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi}) = L_f + \lambda L_g + \gamma |L_f - L_r|$
$g(\boldsymbol{\theta}_g)$
$\hat{z}_1$
$\hat{z}_2$
$\hat{z}_K$
$L_g = \sum_1^K l_g(z_k, \hat{z}_k)$

Fig. 2: The architecture of our AAD framework. The dashed lines indicate the gradient flow direction of back-propagation.

and improve model prediction performance at test time. In practice, the deployed model is often under one particular attack during a given period. We use clean data $\mathcal{D}_c$ and attack data $\mathcal{D}_a$ to simulate possible attack scenarios. We call each attack scenario as a group and assume there are $M+1$ groups, i.e., $\mathcal{D} = \bigcup_i \mathcal{D}^i, i \in [0, M]$. Without loss of generality, we assume $\mathcal{D}^0$ contains only clean data, which represents the scenario without any attack, and each $\mathcal{D}^i$ ($i \in [1, M]$) contains a mixture of clean data and attack examples produced by a particular attack i. Formally, we use $(\boldsymbol{x}_k, y_k, z_k, g_k)$ to denote the k-th data point in $\mathcal{D}$, where $\boldsymbol{x}_k \in \mathcal{X}$ denotes the feature vector, $y_k \in \mathcal{Y}$ denotes the output label of the prediction task, $z_k \in \{0, 1\}$ denotes clean or adversarial, and $g_k \in [0, M]$ denotes the group index. We consider a classification task $\eta_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ from an input $\boldsymbol{x} \in \mathcal{X}$ to an output $y \in \mathcal{Y}$. The goal of the classification model is to obtain the optimal parameters by minimizing the expected loss over test data where the loss function is parameterized by $\theta \times \mathcal{X} \times \mathcal{Y}$.

In this paper, we adopt the widely studied **threat model** of white-box attack that assumes the adversary knows everything related to the basic classification model $\eta_{\boldsymbol{\theta}}$, including training data, model architecture, and hyper-parameters. The adversary seeks to increase error on the test set $\mathcal{D}_{test}$, which corresponds to indiscriminate attack. Our goal is to build an adaptive training model that would be robust against adversarial attacks during test time. In Section V-D1, we further evaluate an advanced threat model that assumes the adversary has full knowledge of our AAD model and the training data.

We assume the training data points are repeatedly generated by the following process: a data distribution $p_{\boldsymbol{xy}} \in \mathcal{P}_{\boldsymbol{xy}}$ is sampled from a set of distributions, and then each data point is sampled from $p_{\boldsymbol{xy}}$. Each $p_{\boldsymbol{xy}}$ then corresponds to a group, i.e., an attack scenario. During the training stage, we organize the labeled training data by groups. During the test stage, there may exist multiple attack scenarios, where each scenario is considered separately and contains unlabeled data (only feature $\boldsymbol{x}$) sampled via a new run of the same generative process. The attack scenarios at test time are likely to be different from those simulated in training, e.g., because of new or unknown attacks. Note that $\mathcal{P}_{\boldsymbol{xy}}$ is large or even infinite.

Our key idea of adversarially adaptive defense is to derive $f_{\boldsymbol{\theta}} : \mathcal{X} \times \mathcal{P}_{\boldsymbol{x}} \to \mathcal{Y}$, indicating that $f_{\boldsymbol{\theta}}$ takes both the input of data $\boldsymbol{x}$ and its marginal distribution $p_{\boldsymbol{x}} \in \mathcal{P}_{\boldsymbol{x}}$. Compared with traditional classification models that usually do not include $\mathcal{P}_{\boldsymbol{x}}$, the adaptive model f uses $p_{\boldsymbol{x}}$ to adapt its prediction on $\boldsymbol{x}$. The underlying assumption of our adaptive model is that $\mathcal{P}_{\boldsymbol{x}}$ can provide extra useful contextual information in addition to $\boldsymbol{x}$ for classification. At the test stage, $f_{\boldsymbol{\theta}}$ first adapts its parameters based on the distribution information $\mathcal{P}_{\boldsymbol{x}}$ of unlabelled data and then produces the predictive results. $\mathcal{P}_{\boldsymbol{x}}$ is only dependent on the feature information itself, which allows $f_{\boldsymbol{\theta}}$ to use the information to improve the prediction from $\mathcal{X} \to \mathcal{Y}$, especially for unseen data or new attacks.

However, it is challenging to learn and then input $\mathcal{P}_{\boldsymbol{x}}$ directly into the model $f_{\boldsymbol{\theta}}$. Similar as [5], we use a batch of data points $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_K)$, where K is the batch size, to derive an empirical distribution $\hat{\mathcal{P}}_{\boldsymbol{x}}$ to approximate $\mathcal{P}_{\boldsymbol{x}}$. Note that the estimation of empirical distribution does not need any label information. We also assume that the unlabelled test data is available to form a batch and so it can be input to $f_{\boldsymbol{\theta}}$. Note that the notion of *batch* is different from the *group* aforementioned. A batch here refers to a training batch containing K data points sampled from a random group to simulate the scenario that different groups contain different attacks (e.g., different distributions). In the next section, we expand the adaptive training to develop our adversarially adaptive defense framework.

B. Framework

The schematic view of the adaptive defense framework is shown in Figure 2. Our framework includes two joint tasks:

prediction and detection. The purpose of the prediction task is to produce classification results for unlabelled data while the purpose of the detection task is to label data clean or adversarial. To achieve each task in the changing environment, we introduce an adaptive context learner h which ingests the features of the data and produces context vectors. A prediction model f and a detection model g then use the learned context vectors, as well as input data, to make predictions. We note that the proposed framework uses an end-to-end training process and the dashed lines in Figure 2 indicate the gradient flow direction of back-propagation.

The adaptive context learner, $h(\cdot;\boldsymbol{\phi}) : \mathcal{X}^K \to \mathcal{C}$ parameterized by $\boldsymbol{\phi}$, takes a batch of K unlabelled data $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_K)$ as input, and outputs the *context vector* $\boldsymbol{c}$. As mentioned previously, the batch data $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_K)$ serves as the empirical approximation from $\hat{\mathcal{P}}_{\boldsymbol{x}}$ to $\mathcal{P}_{\boldsymbol{x}}$. Thus $h(\cdot, \boldsymbol{\phi})$ acts as a mapping function taking $\hat{\mathcal{P}}_{\boldsymbol{x}}$ and producing adaptive data-dependent context vectors. For each example x_k in the batch data, the adaptive context learner works as an encoder module and outputs the context vector c_k. As each batch is sampled from a group, the model takes advantage of context vectors as representations for the hidden distribution within the group. We then conduct an aggregation (e.g., average) on the information from the group to get the context vector $\boldsymbol{c} \leftarrow h(\boldsymbol{x}_1, \dots, \boldsymbol{x}_K; \boldsymbol{\phi})$. For both prediction and detection, $\hat{y}_k \leftarrow f(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_f)$ and $\hat{z}_k \leftarrow g(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_g)$, where $\boldsymbol{x}_k$ and its corresponding context vector $\boldsymbol{c}$ are the input, $\hat{y}_k$ ($\hat{z}_k$) is the output and $\boldsymbol{\theta}_f$ ($\boldsymbol{\theta}_g$) is the parameters of the prediction model f (detection model g).

To improve robustness, we further approximate prediction of $f(\boldsymbol{\theta}_f)$ by using a regularizer $r(\boldsymbol{\theta}_r)$. As shown in Figure 2, for each data point $\boldsymbol{x}_k$, our $h(\boldsymbol{\phi})$ can generate $\boldsymbol{c}_k \in \mathbb{R}^d$ and then derive the context vector of the whole batch by $\boldsymbol{c} = \frac{1}{K}\sum_{k=1}^{K} \boldsymbol{c}_k$. We construct a network $r(\boldsymbol{\theta}_r)$ taking each $\boldsymbol{c}_k$ and obtaining its prediction result $\hat{y}'_k \leftarrow r(\boldsymbol{c}_k; \boldsymbol{\theta}_r)$. During the training process, we guide the outputs $(\hat{y}'_1, \dots, \hat{y}'_K)$ to approximate the prediction result $(\hat{y}_1, \dots, \hat{y}_K)$ produced from $f(\boldsymbol{\theta}_f)$. Specifically we approximate $\hat{y}_k$ and $\hat{y}'_k$ via the cross-entropy loss $l_r = \mathcal{H}(P(\hat{y}_k), P(\hat{y}'_k))$ where $\mathcal{H}$ is the cross-entropy function, $P(\hat{y}_k)$ denotes the softmax output from $f(\boldsymbol{\theta}_f)$, and $P(\hat{y}'_k)$ denotes the softmax output from $r(\boldsymbol{\theta}_r)$ for each data point. Note that $r(\boldsymbol{\theta}_r)$ reconstructs the outcome based on only $(\boldsymbol{c}_1, \cdots, \boldsymbol{c}_K)$ and it does not use input data.

We then compute the joint loss from prediction, detection, and regularization via a multi-task learning framework:

$$\mathcal{L}(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi}) = \sum_{k=1}^{K} l_f(y_k, \hat{y}_k) + \lambda l_g(z_k, \hat{z}_k) + \gamma |l_f - l_r| \tag{1}$$

where l_f is the loss of the prediction, l_g is the loss for the detection, $|l_f - l_r|$ is the l_1 regularization loss, λ and γ are two hyper-parameters used to balance between prediction, detection and regularization losses. We use stochastic gradient descent to optimize and update the parameters:

$$\begin{aligned}(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi}) \leftarrow\ & (\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi}) \\ & - \eta \nabla_{(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})} \mathcal{L}(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})\end{aligned} \tag{2}$$

where η is the step size and $\nabla_{(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})}\mathcal{L}$ denotes the gradient of the joint loss function.

During the test stage, we first derive the context vector $\boldsymbol{c}$ for each new batch of data. For each point $\boldsymbol{x}_k$ in the batch, we then use the prediction model $f(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_f)$ and the detection model $g(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_g)$ to make a prediction and detection. Note that model parameters $\boldsymbol{\theta}_f$, $\boldsymbol{\theta}_g$ and $\boldsymbol{\phi}$ are fixed during test time and the context vector $\boldsymbol{c}$ is only dependent on features of data points in this batch.

Algorithm 1 Adversarially Adaptive Defense (AAD)

```
    /* Training Stage */
Input: training steps $T$, sampling batch size $K$, training data
    $\mathcal{D} = \bigcup_i \mathcal{D}^i$, $i \in [0, M]$, hyper-parameters $\lambda$ and $\gamma$.
Output: adaptation model $h(\boldsymbol{\phi})$, prediction model $f(\boldsymbol{\theta_f})$,
    detection model $g(\boldsymbol{\theta}_g)$.
 1: Initialize $\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g \in \Theta$, $\boldsymbol{\phi} \in \Phi$.
 2: for $t = 1$ to $T$ do
 3:     Sample $i$ uniformly from groups $[0, M]$
 4:     for $k = 1$ to $K$ do
 5:         Sample $(\boldsymbol{x}_k, y_k, z_k)$ uniformly from group $\mathcal{D}^i$
 6:     $\boldsymbol{c} \leftarrow h(\boldsymbol{x}_1, \dots, \boldsymbol{x}_K; \boldsymbol{\phi})$
 7:     for $k = 1$ to $K$ do
 8:         $\hat{y}_k \leftarrow f(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_f)$
 9:         $\hat{z}_k \leftarrow g(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_g)$
10:         $\hat{y}'_k \leftarrow r(\boldsymbol{c}_k; \boldsymbol{\theta}_r)$
11:     Update $(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})$ according to Eq. 2
    /* Test Stage */
Input: $\boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi}$, test batch $\boldsymbol{x}_1, \dots, \boldsymbol{x}_K$.
Output: $\hat{y}_1, \dots, \hat{y}_K$; $\hat{z}_1, \dots, \hat{z}_K$.
12: $\boldsymbol{c} \leftarrow h(\boldsymbol{x}_1, \dots, \boldsymbol{x}_K; \boldsymbol{\phi})$
13: for $k = 1$ to $K$ do
14:     $\hat{y}_k \leftarrow f(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_f)$
15:     $\hat{z}_k \leftarrow g(\boldsymbol{x}_k, \boldsymbol{c}; \boldsymbol{\theta}_g)$
```

Algorithm 1 shows the pseudo code of our algorithm. In line 3-5, we first randomly sample a batch of data $(\boldsymbol{x}_k, y_k, z_k), k \in [1, K]$ from a sampled group $\mathcal{D}^i, i \in [0, M]$. We then learn the context vector $\boldsymbol{c}$ from the feature values of this batch data (line 6). In line 7-10, for each data point $\boldsymbol{x}_k$, we apply the prediction model $f(\boldsymbol{\theta}_f)$, detection model $g(\boldsymbol{\theta}_g)$, and reconstruction model $r(\boldsymbol{\theta}_r)$ to learn the outputs of prediction, detection, and reconstruction. Note that both $f(\boldsymbol{\theta}_f)$ and $g(\boldsymbol{\theta}_g)$ include the context vector $\boldsymbol{c}$ in addition to $\boldsymbol{x}_k$ as input and the regularizer only uses $\boldsymbol{c}_k$ as input. We then compute the joint loss via a multi-task learning framework and use stochastic gradient descent to optimize and update the parameters $\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g$, and $\boldsymbol{\phi}$ (line 11). During test time, we derive the adaptive vector $\boldsymbol{c}$ for the new batch of data (line 12) and use the fixed models to make prediction and detection for each data point (line 14-15).

C. Forming Attack Groups

To successfully develop our adversarially adaptive defense, we need to address one critical challenge of forming representative and informative groups (attack scenarios) in our training. In our framework, we can include adversarial examples produced by both evasion attacks and poisoning attacks. We take advantage of poisoning examples during model training to diversify the attacking groups coming from the attacker and augment the training data. Note that we only include poisoning examples in the training data since poisoning attacks only occur during the training stage. Attacks can also be white-box or black-box where white-box attacks have complete access to the model, including its structure and weights, and black-box attacks can only access the probability or the label of a given input. Theoretically, our AAD can handle both targeted attacks and non-targeted attacks. But in our evaluation, we focus on non-targeted attacks. Table I shows attacks used in our AAD training and testing. We summarize these attacks below and leave the descriptions of each attack as well as their parameter settings used in our evaluation in the supplementary file.

Attack	COMPAS		MNIST		
	Training	Test	Training	Test(S1)	Test(S2)
Boundary Attack (Bond) [18]	✓		✓		
LowProFool (LPF) [19]	✓				
Carlini & Wagner (C&W) [20]	✓				✓
Newton Fool (NF) [21]	✓	✓			✓
Projected Gradient Descent (PGD) [13]	✓	✓	✓		
Influence Attack (Inf) [8]	✓				
Karush-Kuhn-Tucker Attack (KKT) [8]	✓				
Hard Example Attack (HE)	✓				
Label Flipping Attack (LF)	✓				
Fast Sign Gradient Descent (FGSM) [22]		✓	✓	✓	
Deep Fool (DF) [23]		✓	✓		
Jacobian Saliency Map Attack (JSMA) [24]			✓	✓	
Universal Perturbation Attack (UPA) [25]			✓		
HopSkipJump Attack (HSJ) [26]			✓		
Geometric Decision Attack (GDA) [27]			✓		
Frame Saliency Attack (FSA) [28]			✓		
Square Attack (SQA) [29]				✓	✓
Spatial Transformation Attack (STA) [30]				✓	✓

TABLE I: Attacks used in AAD evaluation

Evasion Attacks perturb the test example to obtain a desired classification, leaving the training data and the model unchanged. Roughly speaking, test time attacks, e.g., the L-BFGS attack [31], which was the first method of attacking deep neural network image classifiers, search for a minimally distorted adversarial example $\boldsymbol{x}' = \boldsymbol{x} + \delta$ by adding a perturbation δ to the sample $\boldsymbol{x}$, such that its classifier output $f(\boldsymbol{x}')$ equals to a particular target class t, or $f(\boldsymbol{x}') \neq f(\boldsymbol{x})$ for the non-targeted attack. Advanced and efficient attacks have been further developed. For example, FGSM [22] presents an efficient solution based on the fast gradient sign methodology; PGD [13] is an iterative version of the one-step FGSM, and heuristically searches for the adversarial examples that have the largest loss value in the neighbor of $\boldsymbol{x}$; DeepFool [23] aims to find an optimal path such that $\boldsymbol{x}$ can pass the decision boundary to make classifier $f_{\boldsymbol{\theta}}$ predict a different label. Different from previous attacks which consider one specific victim sample $\boldsymbol{x}$, universal perturbation attack (UPA) [25] tries to find a perturbation δ such that the classifier makes wrong predictions on most of the examples. In addition to the above white-box attacks, we also implement black-box attacks such as boundary attack [18] that generates adversarial examples based solely on observing output labels returned by the targeted model; HopSkipJump attack [26] that is based on the estimate of the gradient direction using binary information at the decision boundary; zeroth order optimization attack [32] that directly uses zeroth-order gradient estimation to craft adversarial examples; and geometric decision based attack [27] that generates query-efficient black-box perturbations.

Poisoning Attacks take place during the training of the machine learning model. An adversary tries to poison the training data by injecting carefully crafted examples to compromise the model performance. Poisoning attacks can be conducted either by direct modification of labels of the training data or by manipulating the input features depending on the adversaries capabilities. Specifically, the attacker observes the test data set $\mathcal{D}_{test}$, as well as a clean training data set $\mathcal{D}_c$, and chooses a fraction of poisoned points $\mathcal{D}_a$ to inject to $\mathcal{D}_c$. The defender observes the combined training data $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_a$, uses a data sanitization defense $\beta = B(\mathcal{D})$ (where B is a function specific to a particular defense) to remove anomalous points, and builds a classification model from the remaining data. Koh et al. [8] proposed three attacks based on three efficient approximations to the bi-level optimization: influence functions, minimax duality, and the Karush-Kuhn-Tucker (KKT) conditions. Influence attack and KKT attack are stronger data poisoning attacks breaking data sanitization defenses and both control the label y and input features $\boldsymbol{x}$ of the poisoned points. The influence attack is a gradient-based attack that iteratively modifies each attack point $(\boldsymbol{x}, y)$ by following the gradient of the test loss with respect to $\boldsymbol{x}$. This attack can also be used at test time as an evasion attack.

IV. Experiment Setting

A. Datasets

We conduct our evaluations on COMPAS [33] and MNIST [34], two widely studied benchmark datasets in adversarial machine learning.

For COMPAS, we use five test time attacks (Bond, LPF, C&W, NF, and PGD) and four poisoning attacks (INF, HE, LF, and KKT) to train our AAD algorithm. During the testing stage, we apply four test time attacks (NF, PGD, FGSM, and DF), two of which (FGSM and DF) are new attacks that are not included in the training data. We first split the original dataset into three parts: clean $\mathcal{D}_c$ (2200), attack candidate $\mathcal{D}_k$ (2000), and test $\mathcal{D}_{test}$ (500). For each attack type, we randomly select 250 examples from $\mathcal{D}_k$ and generate the adversarial examples, 200 of which are used in training and the rest 50 are used in testing. For the training set, we then generate each attack scenario by merging its 200 adversarial examples with 200 clean examples randomly selected from $\mathcal{D}_c$. In total, our training dataset includes ten groups (4000 examples in total) and each group has 400 examples. The first group has 400 clean examples and each of the remaining groups has a combination of 200 clean examples and 200 attack examples. We split the test data into seven groups. For the first three groups, each has 100 clean examples. Each of the remaining groups contains 50 clean examples and 50 attack examples. In total, the test dataset includes 500 clean examples and 200 attack examples (50 for each attack type).

For MNIST, we have two settings: S1 and S2. For both settings, we use the same nine test time attacks (FGSM, PGD, UPA, Bond, DF, JSMA, HSJ, GDA, FSA) to train our AAD algorithm. During the testing stage, for S1 we apply four test time attacks (FGSM, JSMA, SQA, and STA), two of which (SQA and STA) are new attacks and are not included in the training; and for S2 we apply four test time attacks (C&W, NF, SQA, and STA), none of which are included in the training. We split the original dataset (70000) into three parts, clean $\mathcal{D}_c$ (44000), attack candidate $\mathcal{D}_k$ (16000), and test $\mathcal{D}_{test}$ (10000). We follow similar steps as COMPAS to generate attack scenarios for both training and testing. Each attack group has 4000 clean examples and 4000 adversarial examples from one attack. We have ten groups in the training where the first group contains all clean data and each of the remaining groups contains a mixture of clean and adversarial examples. For testing, we have seven groups. The first three are clean groups and each has 2000 clean examples. The next four are attack groups and each contains 1000 clean examples and 1000 attack examples.

B. Defense Baselines

In our experiments, we use the following eight baselines, including three defenses (1)-(3) from adversarial training, one defense (4) from gradient masking, and four defenses (5)-(8) from adversarial example detection.

1) Adversarial Training (AT) [11] is based on ensemble adversarial training. It augments clean training data with crafted adversarial examples from other pre-trained attacking models and then trains a robust prediction model on the augmented data set.

2) Adversarial Training Fast is Better than Free (FAT) [12] uses adversarial examples from weaker and cheaper adversarial attack methods like FGSM. The algorithm can yield models with white-box robustness that is comparable to that obtained with high cost multi-step attacks.

3) Training PGD (ATPGD) [13] uses adversarial examples generated by a PGD attack and then trains the robust model with both clean and crafted examples based on the min-max formulation that casts both attacks and defenses into a common theoretical framework.

4) Defensive Distillation (DD) [14] converts class labels into soft targets (probability vectors instead of hard class labels) which are then used to train an additional model. The advantage of training the second model is to provide a smoother loss function that is more generalized for unknown data and is more robust for adversarial examples.

5) MagNet (MagNet) [15] includes a detector network and a reformer network. The detector learns to separate normal and adversarial examples by approximating the manifold of normal examples and rejects the sample if the distance exceeds a threshold. The reformer network uses auto-encoders to reform an adversarial example to a similar normal example.

6) Activation Defense (AD) [35] uses activations analysis to detect poisoning examples from poisoned dataset. The method tries to get the information from the activation of the last hidden layer of the trained neural network and then applies dimensional reduction and a clustering method for detection.

7) Spectral Signature Defense (SSD) [36] trains a neural network model on a poisoned dataset and then generates a feature representation for each input. They then use singular value decomposition of the covariance matrix of the above representations to compute outlier score for each sample.

8) Feature Squeezing (FS) [16] compares the model's prediction on the original sample with its prediction on the squeezed sample and labels the input as adversarial when the original and squeezed inputs produce different model outputs.

For AT, DD, and AD, we choose the default setting. For FAT, we choose the step size of FGSM attack $\alpha = 0.1$ for COMPAS and keep the default value of 0.3 for MNIST. For ATPGD, we choose the maximum perturbation $\epsilon = 0.1$ for COMPAS and keep the default value of 0.3 for MNIST. For MagNet, we use the default structure from [15] for the detector, reformer, and base model with $drop\_rate = 0.001$. For COMPAS, we use structures of Dense layers for those modules with $drop\_rate = 0.08$. For SSD, we choose the expected percentage of poison $expected\_pp\_poison$ = 0.2 (0.5) and $eps\_multiplier$ = 0.2 (0.15) for COMPAS (MNIST). For FS, we use L_1 distance and the detection threshold = 0.008 (0.002) for COMPAS (MNIST).

C. Implementation Setting

Our AAD algorithm has two hyper-parameters, λ and γ, to balance between prediction, detection and regularization losses as shown in Equation 1. We use $\text{AAD}(\lambda, \gamma)$ to denote our algorithm with chosen values of hyper-parameters. We choose our default setting $\text{AAD}(1, 0.1)$ and simply denote it as AAD. We include $\text{AAD}(1, 0)$, $\text{AAD}(0, 0.1)$ and $\text{AAD}(0, 0)$ in our ablation study as they represent three variants of our adversarially adaptive training, without the use of regularization, detection, or both.

For COMPAS, we choose a two-layer perceptron as the backbone structure for h in our AAD and the size of the context vector c is set the same as the dimension of the input data. For the prediction model f, detection model g, and regularizer r, we choose a logistic regression (LR) model. In both settings of MNIST, we choose two CNN layers (Conv2D + Conv2D) for h and one CNN layer for context vector c with the dimension (channel) as 128. For both f and g, we use two CNN layers followed by two dense layers (Conv2D + Conv2D + Dense + Dense). We use the last two dense layers (Dense + Dense) for regularizer r. In addition to the eight defense baselines shown in Section IV-B, we also include two naive baselines models, B1 and B2. For both of them, we use logistic regression for COMPAS and CNN (Conv2D + Conv2D + Dense + Dense) for MNIST. B1 is trained on only $\mathcal{D}_c$, and B2 has the same training data as our AAD. Note that B1 does not contain a detection model as only clean data is available. For all experiments on COMPAS, we set meta-batch = 3, batch size = 50, training steps = 6000, and learning rate = 0.0003. For MNIST, we choose meta-batch = 3, batch size = 64, training steps = 2000, and learning rate = 0.0003. For all attacks and defense baselines (AT, DD, FS, AD, and SSD), we follow the implementation from IBM Adversarial

Robustness Toolbox v1.2.0 [37]. We use the released code from [12] for defense baselines (FAT, ATPGD), and code from [15] for defense baseline (MagNet).

Metrics. We report the accuracy and F1 score for both prediction and detection. We use macro F1 for prediction on MNIST as it contains multiple output labels. We further report the accuracy and F1 score for each attack scenario. For all of the results, we run our experiments five times with different seeds and report the average values. Due to the page limit, we skip reporting the standard deviation values and instead we summarize comparisons based on the t-test at the end of Section V-A.

V. Experiment Results

A. Overall Performance

Tables II, III, and IV show comparisons of our AAD with all baselines in terms of accuracy and F1 score for both prediction and detection of COMPAS, MNIST S1, and MNIST S2, respectively. Our AAD significantly outperforms all baselines from both prediction and detection perspectives, which demonstrates the power of adaptive training. When comparing the performances of AAD on MNIST S1 and S2, we notice the detection performance of S2 decreases to a 0.65 when S1 obtained a 0.89 in terms of the F1 score, as shown in the last column of Table III and IV. This is understandable as all four attacks in S2 are new attacks. Note that we use N/A to denote those non-applicable results. For example, the three adversarial training based baselines (AT, FAT, and ATPGD) focus on robust training of a prediction model and do not have a detection model. Similarly, the two detection based baselines (AD, SSD) only focus on adversarial sample detection and do not have a prediction model. FS and MagNet are the only two baselines that can both conduct prediction and detection. FS uses feature squeezing and detects adversarial examples by comparing the model's prediction on the original sample with its prediction on the squeezed sample. As demonstrated in our results, FS has the collateral effect of worsening the accuracy of the model, although it helps prevent adversarial attacks. The performance of MagNet also degrades significantly for both detection and prediction because it assumes independence of the adversarial process and only uses normal examples in training.

Significance Testing. We further test the statistical significance of the improvements between our AAD and the baseline models. We run our methods and all baseline models five times, use the independent two-sample t-test, and then calculate the p-value. For both COMPAS and MNIST (both settings), the p-values of testing AAD(1,0.1) against all eight defense baselines and two naive baselines (B1 and B2) are less than 0.01 in terms of both prediction accuracy and detection accuracy.

Execution Time. We report the execution time of AAD and baselines for MNIST S1. All experiments are run on GPU Tesla V100 (32GB RAM) and CPU Xeon 6258R 2.7 GHz. AAD involves adversarial example generation and adaptive training. To generate 5000 adversarial examples for each attack, FGSM, JSMA and PGD take less than one minute; FSA, STA, SQA, DF, UPA, GDA and NF take 4, 7, 9, 10, 15, 38 and 45 minutes, respectively; C&W, HSJ and Bond take 2.5, 3 and 11 hours, respectively. Given the training data, the AAD takes 4 minutes for the training, which is comparable to defense baselines models, e.g., AD, SSD, FS, AT, FAT, and DD take less than one minute; MagNet and ATPGD take 5 and 40 minutes, respectively.

B. Group Level Performance

Tables V, VI, and VII further show group level comparisons of AAD with all baselines for COMPAS, MNIST setting S1, and MNIST setting S2, respectively. We see that for each attack scenario, AAD outperforms all baselines on both prediction and detection tasks with significant margins. Note for MNIST S2, we include four new attacks during test time, including the most challenging C&W attack [20] to test robustness of AAD. The C&W attack, which searches the whole feature space to find a modified point with the lowest loss, is among the most effective attacks [37]. We can see in Table VII AAD achieves high prediction accuracy for all four attack scenarios and high detection accuracy under three attack scenarios of NF, SQA, and STA. The only exception in all our results is the detection accuracy under C&W (50.18%); however, AAD still achieves high prediction accuracy (97.29%).

Method	Prediction		Detection	
	Accuracy (%)	F1 Score	Accuracy (%)	F1 Score
B1	77.43	0.76	N/A	N/A
B2	69.00	0.66	62.71	0.26
AT	69.80	0.18	N/A	N/A
FAT	82.17	0.42	N/A	N/A
ATPGD	76.06	0.24	N/A	N/A
DD	76.88	0.26	N/A	N/A
FS	77.20	0.71	62.20	0.04
MagNet	52.26	0.19	68.03	0.12
AD	N/A	N/A	73.66	0.62
SSD	N/A	N/A	69.51	0.07
AAD	85.43	0.83	85.66	0.72
$\text{AAD}(0, 0)$	83.86	0.81	N/A	N/A
$\text{AAD}(0, 0.1)$	86.91	0.85	N/A	N/A
$\text{AAD}(1, 0)$	82.91	0.80	79.83	0.62

TABLE II: Comparison of AAD and baselines (COMPAS).

Method	Prediction		Detection	
	Accuracy (%)	F1 Score	Accuracy (%)	F1 Score
B1	76.34	0.76	N/A	N/A
B2	81.63	0.80	87.90	0.84
AT	87.19	0.87	N/A	N/A
FAT	89.26	0.89	N/A	N/A
ATPGD	89.05	0.89	N/A	N/A
DD	85.43	0.86	N/A	N/A
FS	84.37	0.85	88.42	0.74
MagNet	75.37	0.81	92.97	0.80
AD	N/A	N/A	71.87	0.33
SSD	N/A	N/A	66.69	0.01
AAD	92.46	0.92	93.91	0.89
$\text{AAD}(0, 0)$	90.48	0.90	N/A	N/A
$\text{AAD}(0, 0.1)$	92.48	0.92	N/A	N/A
$\text{AAD}(1, 0)$	90.61	0.91	90.02	0.82

TABLE III: Comparison of AAD and baselines (MNIST S1).

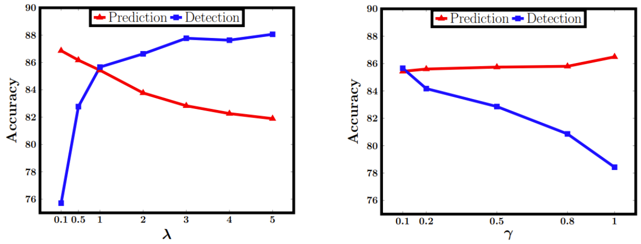
C. Ablation Study and Sensitivity Analysis

Ablation Study. Our AAD framework contains an adaptive context learner, a prediction model, a detection model, and

Method	Prediction		Detection	
	Accuracy (%)	F1 Score	Accuracy (%)	F1 Score
B1	81.60	0.81	N/A	N/A
B2	82.77	0.81	78.54	0.68
AT	87.97	0.88	N/A	N/A
FAT	90.48	0.91	N/A	N/A
ATPGD	90.09	0.90	N/A	N/A
DD	89.62	0.90	N/A	N/A
FS	88.59	0.89	81.83	0.53
MagNet	86.28	0.94	79.56	0.45
AD	N/A	N/A	73.84	0.32
SSD	N/A	N/A	71.33	0.00
AAD	93.29	0.93	84.04	0.65
$\text{AAD}(0, 0)$	91.34	0.91	N/A	N/A
$\text{AAD}(0, 0.1)$	93.31	0.93	N/A	N/A
$\text{AAD}(1, 0)$	91.57	0.91	80.70	0.59

TABLE IV: Comparison of AAD and baselines (MNIST S2).

a regularizer. In our ablation study, we include three variants $\text{AAD}(1, 0)$, $\text{AAD}(0, 0.1)$ and $\text{AAD}(0, 0)$, representing the cases where regularization and/or detection are not used. We report their results in the last block of Tables II-VII. Note that $\text{AAD}(0, 0)$ only includes adaptive prediction and $\text{AAD}(0, 0.1)$ further adds reconstruction regularization, therefore their detection results are shown as N/A in the tables. Comparing the prediction output of $\text{AAD}(0, 0)$ with those baselines, we can see our adaptive training itself (without regularizer and detection) already significantly improves the prediction accuracy. Comparing $\text{AAD}(0, 0.1)$ with $\text{AAD}(0, 0)$, we can see the usefulness of our regularizer. We emphasize again that our AAD algorithm achieves the best performance when all components are used in the training.



(a) Sensitivity analysis of λ, $\gamma = 0.1$.

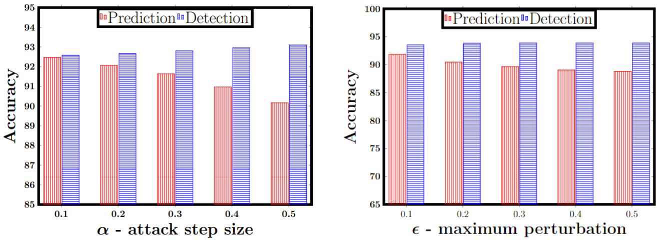
(b) Sensitivity analysis of γ, $\lambda = 1$.

Fig. 3: Sensitivity analysis of parameters λ and γ (COMPAS).

Sensitivity Analysis. Figure 3 shows the sensitivity analysis of parameters λ and γ of AAD over COMPAS. As shown in Figure 3a, when we fix $\gamma = 0.1$ and increase λ, the detection accuracy increases and the prediction accuracy decreases slightly. This is because a large value of λ indicates more weight on detection in the joint loss. Similarly, as shown in Figure 3b, when we fix $\lambda = 1$ and increase γ, the detection accuracy decreases whereas the prediction accuracy increases. This is because our regularizer aims to reconstruct prediction based on the context information.

D. AAD under Full Knowledge Attack

For all previous results, adversarial examples included in training are produced by attacks based on the basic prediction model η, and the adversary also generates test time adversarial examples based on η. This corresponds to the scenario where the adversary thought the baseline B1 is being deployed. Our results reported in Sections V-A-V-C have demonstrated that our trained AAD model, if deployed, would be robust against those attacks. One interesting question is how robust AAD would be if the adversary knows AAD's deployment and generates adversarial examples based on the full knowledge of AAD training process and training data. To answer this question, we conduct another experiment by enabling the FGSM attack and PGD attack to have all the above knowledge.



(a) Full Knowledge FGSM attack

(b) Full Knowledge PGD attack

Fig. 4: Full Knowledge FGSM/PGD attack on AAD (MNIST S1).

1) AAD under Full Knowledge FGSM Attack: FGSM generates adversarial examples based on $\boldsymbol{x}' = \boldsymbol{x} + \alpha \cdot sign(\nabla_{\boldsymbol{x}} \mathcal{L}(\theta, \boldsymbol{x}, y))$ for non-target attack where $\nabla_{\boldsymbol{x}}$ denotes the gradient of the model with respect to a normal sample $\boldsymbol{x}$ with correct label y, and α is the step size. Replacing $\mathcal{L}$ with the joint loss $\mathcal{L}(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})$ shown in Equation 1, we can generate new adversarial examples against AAD. Figure 4a shows the prediction and detection results for setting MNIST S1 under the FGSM attack with five step size values. We can see AAD still achieves high prediction accuracy and detection accuracy. Moreover, when step size α increases, prediction accuracy decreases while detection accuracy increases. This is because a large α allows a larger distortion or perturbation when generating adversarial examples.

2) AAD under Full Knowledge PGD Attack: Similar to FGSM attack, we generate adversarial examples using PGD L_∞ attack with full knowledge of the joint loss $\mathcal{L}(\boldsymbol{\theta}_r, \boldsymbol{\theta}_f, \boldsymbol{\theta}_g, \boldsymbol{\phi})$. We use setting MNIST S1 with the replacement of FGSM with PGD in the testing stage. Figure 4b shows the results for five maximum perturbation values ϵ where the attack step size α is configured as 0.1 to adapt to our model. Note that PGD is an extension of FGSM using an iterative method to generate stronger adversarial examples. When ϵ increases, prediction accuracy decreases while detection accuracy increases because more perturbation is allowed. This trend is also consistent with our observation from Figure 4a in Section V-D1. Overall, ADD model is still robust under white-box attacks.

E. Robustness of AAD when One Test Group Contains Multiple Attack Types

In this section, we demonstrate the effectiveness of *context vector* $\boldsymbol{c}$ defined in Section III-B in response to a remarkable

Method	Prediction Accuracy (%)					Detection Accuracy (%)				
	Clean Test data	NF	PGD	FGSM	DF	Clean Test data	NF	PGD	FGSM	DF
B1	90.73	74.40	61.20	63.60	70.60	N/A	N/A	N/A	N/A	N/A
B2	81.06	62.00	59.00	59.20	61.20	78.06	51.80	49.00	51.60	52.40
AT	81.13	56.00	61.40	64.00	63.80	N/A	N/A	N/A	N/A	N/A
FAT	96.40	78.60	62.00	68.40	77.00	N/A	N/A	N/A	N/A	N/A
ATPGD	90.60	61.80	62.40	67.40	69.00	N/A	N/A	N/A	N/A	N/A
DD	91.80	68.40	64.20	61.80	68.40	N/A	N/A	N/A	N/A	N/A
FS	92.14	69.00	64.40	61.20	69.40	86.20	44.20	45.60	44.40	42.60
MagNet	50.87	56.20	53.80	54.00	49.20	92.53	46.80	49.60	48.20	54.00
AD	N/A	N/A	N/A	N/A	N/A	79.60	74.00	68.40	63.20	71.20
SSD	N/A	N/A	N/A	N/A	N/A	95.80	48.60	49.60	49.80	52.00
AAD	97.33	74.00	83.60	74.60	73.80	95.47	90.20	68.80	73.20	81.00
$\text{AAD}(0,0)$	96.47	71.80	72.20	79.60	74.00	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(0,0.1)$	97.13	77.40	89.20	77.20	72.80	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(1,0)$	96.20	72.20	73.60	71.60	74.40	91.20	87.20	60.80	62.60	74.60

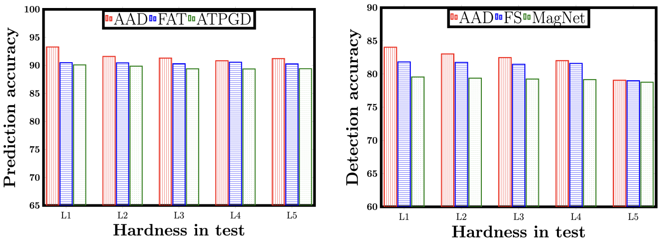
TABLE V: Comparison of AAD and baselines on each group (COMPAS).

Method	Prediction Accuracy (%)					Detection Accuracy (%)				
	Clean Test data	FGSM	JSMA	SQA	STA	Clean Test data	FGSM	JSMA	SQA	STA
B1	92.24	59.97	80.27	56.29	61.10	N/A	N/A	N/A	N/A	N/A
B2	90.87	81.44	85.42	72.07	59.89	93.95	98.16	76.82	97.65	60.81
AT	96.32	87.13	92.38	91.20	50.64	N/A	N/A	N/A	N/A	N/A
FAT	98.80	90.08	96.58	83.27	58.52	N/A	N/A	N/A	N/A	N/A
ATPGD	98.66	91.41	96.75	77.96	61.25	N/A	N/A	N/A	N/A	N/A
DD	98.23	77.46	91.15	79.21	55.49	N/A	N/A	N/A	N/A	N/A
FS	98.25	74.42	92.06	91.78	57.62	100.00	88.07	65.35	88.69	76.83
MagNet	99.22	49.36	66.34	49.68	64.56	99.82	99.93	82.25	99.59	52.38
AD	N/A	N/A	N/A	N/A	N/A	92.23	50.90	50.21	50.04	79.61
SSD	N/A	N/A	N/A	N/A	N/A	92.35	47.17	47.69	48.69	47.23
AAD	98.29	92.57	96.05	87.69	76.03	95.41	99.99	93.47	99.37	78.28
$\text{AAD}(0,0)$	97.44	90.75	94.09	85.49	70.72	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(0,0.1)$	98.21	92.37	96.04	87.38	76.84	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(1,0)$	97.29	90.27	94.02	85.05	73.04	91.43	100.00	86.84	99.28	69.71

TABLE VI: Comparison of AAD and baselines on each group (MNIST S1).

Method	Prediction Accuracy (%)					Detection Accuracy (%)				
	Clean Test data	C&W	NF	SQA	STA	Clean Test data	C&W	NF	SQA	STA
B1	92.24	89.14	87.96	56.29	61.10	N/A	N/A	N/A	N/A	N/A
B2	90.97	88.25	86.60	72.07	59.89	93.95	50.50	58.97	97.65	60.81
AT	96.01	93.80	92.70	90.40	50.83	N/A	N/A	N/A	N/A	N/A
FAT	98.78	97.85	97.63	82.87	58.68	N/A	N/A	N/A	N/A	N/A
ATPGD	98.72	97.72	97.59	77.65	61.48	N/A	N/A	N/A	N/A	N/A
DD	98.73	97.75	97.47	78.03	57.88	N/A	N/A	N/A	N/A	N/A
FS	98.47	97.39	96.41	73.58	57.38	100	51.17	56.68	86.95	77.72
MagNet	99.22	98.76	93.34	49.68	64.56	99.82	50.01	55.44	99.59	52.38
AD	N/A	N/A	N/A	N/A	N/A	74.04	73.72	73.51	73.66	73.89
SSD	N/A	N/A	N/A	N/A	N/A	71.59	71.30	70.83	71.01	71.40
AAD	98.29	97.29	97.02	87.69	76.03	95.41	50.18	74.12	99.37	78.28
$\text{AAD}(0,0)$	97.44	95.74	95.09	85.49	70.72	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(0,0.1)$	98.21	97.31	96.94	87.38	76.93	N/A	N/A	N/A	N/A	N/A
$\text{AAD}(1,0)$	97.29	95.71	95.28	85.05	73.04	91.43	50.33	71.30	99.28	69.71

TABLE VII: Comparison of AAD and baselines on each group (MNIST S2).



(a) Prediction robustness

(b) Detection robustness

Fig. 5: AAD under hard test scenario (MNIST S2).

question of how effective the context vector would be in capturing and presenting group contextual information such that AAD can adapt to new test time attacks. Note that previous results are conducted with the setting that each attack group in test data contains only one type of attack. In real-world situations, the adversary can include multiple attacks in each group to make the defense fail to detect them. Therefore, to answer the question of whether AAD could be robust under challenging test scenarios, we design five test scenarios **L1** → **L5** using setting MNIST S2 with increasing difficulty in test attack groups. In **L1**, each attack group contains one attack type (this is the default test scenario for setting MNIST S2). For **L2**, **L3**, and **L4**, each attack group contains two, three, and four types of attacks with the same ratio. For **L5**, each attack group contains a random number of attacks with random ratios. For the training, we still use the same training scenario as setting MNIST S2, which includes one attack in each training attack group. For a comprehensive comparison, we also choose the top two baseline performers from the prediction task and the detection task. As shown in Table

IV, FAT and ATPGD are the two baselines with the best performance for the prediction task. For the detection task, FS and MagNet are the best two performers. Figures 5a and 5b show the prediction comparison and the detection comparison under five test scenarios, respectively. The prediction accuracy and detection accuracy of all defensive models decrease when the difficulty level increases from **L1** to **L5**, which is a typical trend. Our AAD model achieves the best performance in all test scenarios in terms of both prediction and detection accuracy. These results explicitly demonstrate the robustness of our AAD model with new test time attacks and also reveal that the context vector can certainly help learn contextual information in different test environments.

VI. Conclusions and Future Work

We have presented a novel defense framework that adapts at test time to unknown attacks. Models trained with our adversarially adaptive defense are less sensitive to adversarial examples, and are thus more suitable for deployment in sensitive security settings. Our comprehensive empirical evaluations showed the promising results of our AAD's robustness and adaption against a variety of new attacks. In future work, we will continue the study of our approach's robustness when the adversary has full knowledge of AAD's training process and training data. We will also extend our approach to other attacks, e.g., the backdoor attack [38] that injects the chosen backdoor pattern onto particular training examples from a given class, and does not affect a model's behavior on typical, benign data.

Reproducibility. All source code and datasets can be downloaded at https://github.com/minhhao97vn/AAD.

Acknowledgement

This work was supported in part by NSF 1564250, 1946391, and 2147375.

References

[1] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Networks Learn. Syst.*, 2019.

[2] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "A survey on adversarial attacks and defences," *CAAI Trans. Intell. Technol.*, 2021.

[3] H. Xu, Y. Ma, H. Liu, D. Deb, H. Liu, J. Tang, and A. K. Jain, "Adversarial attacks and defenses in images, graphs and text: A review," *IJAC*, 2020.

[4] N. Papernot, P. D. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *EuroS&P*, 2018.

[5] M. Zhang, H. Marklund, N. Dhawan, A. Gupta, S. Levine, and C. Finn, "Adaptive risk minimization: Learning to adapt to domain shift," in *NeurIPS*, 2021.

[6] G. Blanchard, A. A. Deshmukh, Ü. Dogan, G. Lee, and C. Scott, "Domain generalization by marginal transfer learning," *J. Mach. Learn. Res.*, 2021.

[7] D. Wang, E. Shelhamer, S. Liu, B. A. Olshausen, and T. Darrell, "Tent: Fully test-time adaptation by entropy minimization," in *ICLR*, 2021.

[8] P. W. Koh, J. Steinhardt, and P. Liang, "Stronger data poisoning attacks break data sanitization defenses," *Mach. Learn.*, vol. 111, no. 1, 2022.

[9] L. Muñoz-González, B. Pfitzner, M. Russo, J. Carnerero-Cano, and E. C. Lupu, "Poisoning attacks with generative adversarial nets," *CoRR*, vol. abs/1906.07773, 2019.

[10] B. Zhao and Y. Lao, "Towards class-oriented poisoning attacks against neural networks," in *WACV*, 2022.

[11] F. Tramèr, A. Kurakin, N. Papernot, I. J. Goodfellow, D. Boneh, and P. D. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *ICLR (Poster)*, 2018.

[12] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," in *ICLR*, 2020.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR (Poster)*, 2018.

[14] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *IEEE S&P*, 2016.

[15] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *CCS*, 2017.

[16] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *NDSS*, 2018.

[17] J. Chen, Y. Guo, X. Wu, T. Li, Q. Lao, Y. Liang, and S. Jha, "Towards adversarial robustness via transductive learning," *CoRR*, vol. abs/2106.08387, 2021.

[18] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," in *ICLR (Poster)*, 2018.

[19] V. Ballet, X. Renard, J. Aigrain, T. Laugel, P. Frossard, and M. Detyniecki, "Imperceptible adversarial attacks on tabular data," *CoRR*, vol. abs/1911.03274, 2019.

[20] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE S&P*, 2017.

[21] U. Jang, X. Wu, and S. Jha, "Objective metrics and gradient descent algorithms for adversarial examples in machine learning," in *ACSAC*, 2017.

[22] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR (Poster)*, 2015.

[23] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *CVPR*, 2016.

[24] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *EuroS&P*, 2016.

[25] S. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *CVPR*, 2017.

[26] J. Chen, M. I. Jordan, and M. J. Wainwright, "Hopskipjumpattack: A query-efficient decision-based attack," in *IEEE S&P*, 2020.

[27] A. Rahmati, S. Moosavi-Dezfooli, P. Frossard, and H. Dai, "Geoda: A geometric framework for black-box adversarial attacks," in *CVPR*, 2020.

[28] N. Inkawhich, M. Inkawhich, Y. Chen, and H. Li, "Adversarial attacks for optical flow-based action recognition classifiers," *CoRR*, vol. abs/1811.11875, 2018.

[29] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, "Square attack: A query-efficient black-box adversarial attack via random search," in *ECCV (23)*, 2020.

[30] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *ICML*, 2019.

[31] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *ICLR (Poster)*, 2014.

[32] P. Chen, H. Zhang, Y. Sharma, J. Yi, and C. Hsieh, "ZOO: zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *AISec@CCS*, 2017.

[33] J. Larson, S. Mattu, L. Kirchner, and J. Angwin, "Compas dataset," https://github.com/propublica/compas-analysis, 2017.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, 1998.

[35] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. M. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," in *SafeAI@AAAI*, 2019.

[36] B. Tran, J. Li, and A. Madry, "Spectral signatures in backdoor attacks," in *NeurIPS*, 2018.

[37] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy, and B. Edwards, "Adversarial robustness toolbox v1.2.0," *CoRR*, vol. 1807.01069, 2018.

[38] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning," *CoRR*, vol. abs/1712.05526, 2017.