Reproducibility for paper: Defending Evasion Attacks via Adversarially Adaptive Training

I. Reproducibility

In this section, we describe the reproducibility of the paper.
Models and algorithms. For our AAD model and other algorithms used in the paper, we have a clear description of the mathematical setting, algorithm, network architectures, and parameter settings for all experiments. We report the execution time for each attack, defense, and ADD model for efficiency comparison.
Datasets. For all datasets used, we report the statistics of the number of examples, train/test splits, and pre-processing steps. We also include all data files used for conducting experiments in the source code folder.
Source code. All details related to source code execution, commands and dependencies are clearly described in README files. The trained models can be saved and loaded for later evaluations.
Experimental results. The hyper-parameter settings for reproducing the results are reported in the paper and README files. We also include a description of the computing infrastructure used in the experiments.

II. Implementation

We conduct the experiments on Ubuntu 20.04.3 LTS with GPU Tesla V100 (32GB RAM) and CPU Xeon 6258R 2.7 GHz. The implementation is based on Python 3.7 on Conda environments.

A. Implementation of Attacks

The folder **Attack_COMPAS** and **Attack_MNIST** in this source code includes the implementation of attacks used in experiments for COMPAS and MNIST dataset, respectively. Note that this part of code is built using implementation from IBM Adversarial Robustness Toolbox [1] (https://github.com/Trusted-AI/adversarial-robustness-toolbox)and the paper [2] (https://github.com/locuslab/fast_adversarial/). Please refer to **README.MD** file in this folder for the detailed instruction about requirements, commands and executions.

We generate adversarial examples from 18 attacks in the paper from feasible attack data created from the original COMPAS and MNIST. The setting and description for attacks are shown in III.

B. Implementation of AAD

The folder **AAD** in this source code includes the implementation of Adversarially Adaptive Defense. Please refer to **README.MD** file in this folder for detailed instructions about requirements, commands, and executions.

This folder has all the data files used in the experiments from the paper. The folder **data** includes clean examples from the original dataset and adversarial examples generated by adversarial attacks, as mentioned in the previous section. The code is used to train and validate the AAD model and simple baselines under the settings and scenarios in the paper.

C. Implementation of Defense Baselines

The folder **Defense_COMPAS** and **Defense_MNIST** in this source code include the implementation of defenses used in experiments for COMPAS and MNIST, respectively. Note that this part of the code is built using the implementation from IBM Adversarial Robustness Toolbox [1] (https://github.com/Trusted-AI/adversarial-robustness-toolbox). Please refer to **README.MD** file in this folder for detailed instructions about requirements, commands, and executions.

Moreover, the folder **MagNet_COMPAS** and **MagNet_MNIST** in this source code includes the implementation of MagNet defense used in experiments for COMPAS and MNIST dataset, respectively. Note that this part of code is built using implementation from [3] (https://github.com/Trevillie/MagNet)

The results from baselines used for comparison in the paper are conducted using this code. The setting and description of defenses are reported in the paper.

III. Attacking Models

To generate adversarial examples, we use a simple NN (Dense + Dense) structure as pretrained classifier for COMPAS and a CNN (Conv2D + Conv2D + Dense + Dense) structure as pretrained classifier for MNIST. We focus on non-target attacks and mainly follow the default attack settings from IBM Adversarial Robustness Toolbox v1.2.0 [1] and from the released code of [4] for both COMPAS and MNIST. In this section, we briefly describe each attack and specify parameter settings used in the evaluation. For those attacks that we use the default parameter setting from the original software, we skip reporting them.

A. Test Time Evasion Attacks

Fast Gradient Sign Method (FGSM). Goodfellow et al. [5] propose an efficient attacking method based on the fast gradient sign methodology:

$$\boldsymbol{x}' = \boldsymbol{x} + \alpha \cdot sign(\nabla_{\boldsymbol{x}} \mathcal{L}(\theta, \boldsymbol{x}, y)) \quad \text{non-target} \tag{1a}$$

$$\boldsymbol{x}' = \boldsymbol{x} - \alpha \cdot sign(\nabla_{\boldsymbol{x}} \mathcal{L}(\theta, \boldsymbol{x}, t)) \quad \text{target on } t \tag{1b}$$

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. In our experiments, we choose the step size $\alpha = 0.1$ for COMPAS and use the default value of 0.3 for MNIST.

Projected Gradient Descent (PGD). Madry et al. [6] introduce an iterative version of the one-step FGSM and its formulation of non-targeted attack is:

$$\begin{aligned} &\boldsymbol{x}^0 = \boldsymbol{x} \\ &\boldsymbol{x}^t = Clip(\boldsymbol{x}^{t-1} + \alpha \cdot sign(\nabla_{\boldsymbol{x}} \mathcal{L}(\theta, \boldsymbol{x}^{t-1}, y))) \end{aligned} \tag{2}$$

where $Clip$ denotes the function to project the modified point within the range of the ϵ-neighbor ball of $\boldsymbol{x}$. The PGD attack heuristically searches the most-adversarial examples that have the largest loss value in the neighbor ball of $\boldsymbol{x}$.

Carlini and Wagner's Attack (C&W). C&W's attack [7] aims to solve the following optimization problem:

$$\begin{aligned} &\text{minimize} \;\; \mathcal{D}(\boldsymbol{x}, \boldsymbol{x} + \delta) + c\mathcal{L}(\boldsymbol{x} + \delta, t) \\ &\text{subject to} \;\; \boldsymbol{x} + \delta \in [0, 1]^n \end{aligned} \tag{3}$$

where $\mathcal{D}$ is some distance metric , c is a chosen constant, and $[0, 1]^n$ refers to the space of all n-length vectors consisting of real numbers between 0 and 1 inclusive. Minimization of $\mathcal{L}(\boldsymbol{x} + \delta, t)$ encourages the attack algorithm to find a modified point that has the lowest loss for label t than any other label. Thus the classifier will be fooled to predict the modified point as class t.

DeepFool (DF). Moosavi-Dezfooli et al. [8] propose DF by studying the decision boundary of classifier $f_{\boldsymbol{\theta}}$ around data point $\boldsymbol{x}$. Its goal is to find an optimal path such that $\boldsymbol{x}$ can pass the decision boundary to make classifier $f_{\boldsymbol{\theta}}$ predict a different class label for $\boldsymbol{x}$.

Jacobian-based Saliency Map Attack (JSMA). Papernot et al. [9] introduce the greedy attack algorithm based on calculating the Jacobian matrix of the score function to iteratively manipulate the pixel which is the most influential to the model output. In our experiments, we choose smaller value for the maximum fraction of features being perturbed $\gamma = 0.15$ (default 1) for MNIST.

Newton Fool (NF). Jiang et al. [10] develop an untargeted attack that tries to decrease the probability of the original class by performing gradient descent. The NF is similar to DeepFool and further introduces computer-vision algorithms such as edge detectors as a way for evaluating the quality adversarial examples.

Universal Perturbation Attack (UPA). Moosavi-Dezfooli et al. [11] introduce the universal attack. Different from attacks which consider one specific victim sample $\boldsymbol{x}$, UPA tries to find a perturbation δ such that the classifier makes wrong predictions on most of the examples:

$$\begin{aligned} &\| \delta \|_p \leq \epsilon \\ &\mathop{\mathbb{P}}_{\boldsymbol{x} \sim \mu}(f(\boldsymbol{x} + \delta) \neq f(\boldsymbol{x})) \geq 1 - \sigma \end{aligned} \tag{4}$$

where $\| \delta \|_p$ is the L_p norm with $p \in [1, \infty)$, ϵ controls the magnitude of the perturbation vector δ, and σ indicates the desired fool rate of all data sampled from the distribution μ. In our experiments, we choose higher value for the magnitude of the perturbation $\epsilon = 0.3$ (default 0.1) for MNIST.

LowProFool (LPF). Ballet et al. [12] propose LowProFool to generate adversarial examples for tabular data. The problem formulation is similar as FGSM:

$$\begin{aligned} &\text{minimize} \;\; d(\delta) \\ &\text{subject to} \;\; f(\boldsymbol{x} + \delta) \neq f(\boldsymbol{x}), \boldsymbol{x} + \delta \in A \end{aligned} \tag{5}$$

where $\boldsymbol{x} + \delta \in A$ requires the generated point to be feasible that respects the discrete, categorical and continuous features, and $d(\delta)$ is defined as the L_p norm $d(\delta) = \| \delta \odot v \|_p$ where v is a vector containing the importance coefficient of each feature in $\boldsymbol{x}$. The introduction of v can improve the attack success rate over FGSM by putting higher weights on more relevant features. In our experiments, we choose smaller value for the rate of updating the perturbation vector $\eta = 0.1$ (default 0.2) and its decrease after each step $\eta_{decay} = 0.05$ (default 0.98) for COMPAS.

All the above attacks are white-box attacks and all below are black-box attacks.

Boundary Attack (Bond). Brendel et al. [13] propose boundary attack as one decision-based adversarial attack. The attack generates adversarial examples based solely on observing output labels returned by the targeted model. The attack is based on rejective sampling and generates the adversarial example by iteratively walking through the boundary between the adversarial and the non-adversarial region. It defines the adversarial criterion function $c(\cdot)$, a suitable proposal distribution $\mathcal{P}$ and an initial adversarial data point $\tilde{\boldsymbol{x}}$. Then it performs the following step iteratively:

$$\begin{aligned} &\delta^t \sim \mathcal{P}(\tilde{\boldsymbol{x}}^{t-1}) \\ &\tilde{\boldsymbol{x}}^t = \tilde{\boldsymbol{x}}^{t-1} + \delta^t \end{aligned} \tag{6}$$

where the process of Equation 6 will be stopped if $\tilde{\boldsymbol{x}}^t$ does not satisfy the adversarial criterion function $c(\cdot)$. δ^t is a small perturbation at each step so that the generated adversarial example is close to the targeted non-adversarial data.

HopSkipJump Attack (HSJ). Chen et al. [14] develop an attack that is based on the estimate of the gradient direction using binary information at the decision boundary. This attack requires fewer model queries than other decision-based adversarial attacks like Bond and achieves competitive performance in attacking several widely-used defense mechanisms.

Zeroth Order Optimization Attack (ZOO). Chen et al. [15] propose a score-based black-box attack that directly use zeroth-order gradient estimation to craft adversarial examples. ZOO captures the gradient information around a target point

$\boldsymbol{x}$ by observing the variations of the prediction probability of classifier f_θ when the feature values of $\boldsymbol{x}$ are changed. For the ith feature of $\boldsymbol{x}$, ZOO adds or subtracts $\boldsymbol{x}^i$ by a small value δ, and estimates the gradient information from the output of f_θ as:

$$\frac{\partial f_\theta(\boldsymbol{x}^i)}{\partial \boldsymbol{x}^i} \approx \frac{f_\theta(\boldsymbol{x}^i + \delta) - f_\theta(\boldsymbol{x}^i - \delta)}{2\delta} \tag{7}$$

With the approximate gradient information, we can apply previous attack approach such as FGSM, C&W attack approach to generate adversarial examples.

Frame Saliency Attack (FSA). Inkawhich et al. [16] develop a untargeted adversarial attack for action recognition systems in both white-box and black-box settings. The attack combines the gradients of a differentiable optical flow calculation algorithm and a convolutional neural network to perturb the video frames themselves. Using sparsely perturbed examples, the attack can only require a single frame perturbation.

Square Attack (SQA). Andriushchenko et al. [17] develop a score-based black-box attack. The square attack does not depend on local gradient information and thus is not affected by gradient masking based defense. The square attack selects localized square shaped updates at random positions such that the perturbation at each iteration is close to the boundary of the feasible set. The attack is query efficient and achieves good success rate especially in the untargeted setting. In our experiments, we choose higher value of $nb\_restarts = 50$ (default 1) for MNIST.

Spatial Transformation Attack (STA). Engstrom et al. [18] studied the neural network based classifier robustness to image translations and rotations. They show that a small number of randomly chosen perturbations of the input are sufficient to degrade the classifier's performance. In our experiments, we choose different values for the maximum translation = 10 (default 0), the number of translations = 3 (default 1), the maximum rotation = 18 (default 0), and the number of rotations = 3 (default 1) for MNIST.

Geometric Decision Based Attack (GDA). Rahmati et al. [19] develop a geometric framework to generate adversarial examples in the black-box setting where the adversary can only apply a small number of queries, each of them returning the top-1 label of the classifier. They propose an effective iterative algorithm to generate query-efficient black-box perturbations with small l_p norms.

B. Poisoning Attacks

Poisoning attack takes place during the training phase of the machine learning model. An adversary tries to poison the training data by injecting carefully crafted examples to compromise the model performance. Koh et al. [4] developed three attacks that can bypass data sanitization defenses. The ideas are to place poisoned examples near one another and formulate each attack as a constrained optimization problem where the attacker's objective is to maximize the test loss of the model that the defender learns on the mixture of the clean and poisoned data and constraints are designed to ensure that the poisoned points evade detection. Specifically, the attacker observes the test data $\mathcal{D}_{test}$ as well as a clean training data set $\mathcal{D}_c = \{(x_i, y_i)\}_{i=1}^n$, and chooses ϵn poisoned points $\mathcal{D}_p$ to inject to $\mathcal{D}_c$. The defender observes the combined training data $\mathcal{D} = \mathcal{D}_c \cup \mathcal{D}_p$, 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 classification model from the remaining data.

Influence Attack (INF). The influence attack performs gradient ascent on each sample $(\boldsymbol{x}, y) \in \mathcal{D}_p$ to maximize the test loss.

$$\boldsymbol{x}' = \boldsymbol{x} - \eta g_{\hat{\theta}, \mathcal{D}_{test}}^\top H_{\hat{\theta}}^{-1} \frac{\partial^2 l(\hat{\theta}; \boldsymbol{x}, y)}{\partial \theta \partial \boldsymbol{x}} \quad \text{gradient update} \tag{8a}$$

$$\boldsymbol{x}' = argmin_{x \in \mathcal{F}_\beta} ||\boldsymbol{x} - \boldsymbol{x}'||_2 \quad \text{projection} \tag{8b}$$

where $H_{\hat{\theta}}$ is the Hessian of the training loss at $\hat{\theta}$. To maintain the feasibility of each poisoned point in $\mathcal{D}_p$, the projection step is to project $(\boldsymbol{x}', y)$ onto the feasible set $\mathcal{F}_\beta$ by setting it to the feasible point $(\boldsymbol{x}, y) \in \mathcal{F}_\beta$ that is closest in l_2 distance to $(\boldsymbol{x}', y)$.

KKT Attack (KKT). The KKT attack further uses fast heuristics to find decoy parameters that we want the defender to learn, and find poisoned data $\mathcal{D}_p$ such that the defender is tricked into learning the decoy parameters. The min-max attack relies on the same decoy parameters in the KKT attack but assumes that the clean data and the test data are drawn from the same distribution (i.e., the attacker is performing an indiscriminate attack).

The influence attack that we presented in Section III-B 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 in test time as evasion attack. Both the influence and KKT attack can be used as indiscriminate and targeted attacks as they do not make any assumptions on the nature of the test data. In contrast, the min-max attack assumes that the training error is a good approximation to the test error, and thus is only appropriate in the indiscriminate attack setting.

Hard Example Attack (HE). Attacker randomly selects data examples from hard examples set.

Label Flipping Attack (LF). Attacker randomly selects data examples from the candidate attack data $\mathcal{D}_k$ and flips their labels.

REFERENCES

[1] 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.

[2] E. Wong, L. Rice, and J. Z. Kolter, "Fast is better than free: Revisiting adversarial training," *arXiv preprint arXiv:2001.03994*, 2020.

[3] D. Meng and H. Chen, "Magnet: A two-pronged defense against adversarial examples," in *CCS 2017*. ACM, 2017, pp. 135–147.

[4] P. W. Koh, J. Steinhardt, and P. Liang, "Stronger data poisoning attacks break data sanitization defenses," *arXiv preprint arXiv:1811.00741*, 2018.

[5] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[6] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *IEEE S&P 2017*. IEEE, 2017, pp. 39–57.

[8] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *CVPR 2016*, 2016, pp. 2574–2582.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *EuroS&P 2016*. IEEE, 2016, pp. 372–387.

[10] U. Jang, X. Wu, and S. Jha, "Objective metrics and gradient descent algorithms for adversarial examples in machine learning," in *ACSAC 2017*, 2017, pp. 262–277.

[11] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *CVPR 2017*, 2017, pp. 1765–1773.

[12] V. Ballet, X. Renard, J. Aigrain, T. Laugel, P. Frossard, and M. Detyniecki, "Imperceptible adversarial attacks on tabular data," *arXiv preprint arXiv:1911.03274*, 2019.

[13] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," *arXiv preprint arXiv:1712.04248*, 2017.

[14] J. Chen, M. I. Jordan, and M. J. Wainwright, "Hopskipjumpattack: A query-efficient decision-based attack," in *S&P 2020*. IEEE, 2020, pp. 1277–1294.

[15] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM workshop on artificial intelligence and security*, 2017, pp. 15–26.

[16] N. Inkawhich, M. Inkawhich, Y. Chen, and H. Li, "Adversarial attacks for optical flow-based action recognition classifiers," *arXiv preprint arXiv:1811.11875*, 2018.

[17] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein, "Square attack: a query-efficient black-box adversarial attack via random search," in *ECCV*. Springer, 2020, pp. 484–501.

[18] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the landscape of spatial robustness," in *ICML 2019*. PMLR, 2019, pp. 1802–1811.

[19] A. Rahmati, S.-M. Moosavi-Dezfooli, P. Frossard, and H. Dai, "Geoda: a geometric framework for black-box adversarial attacks," in *CVPR 2020*, 2020, pp. 8446–8455.