Game Theoretical Adversarial Deep Learning with Variational Adversaries

Aneesh Sreevallabh Chivukula, Xinghao Yang, Wei Liu* *Member, IEEE*, Tianqing Zhu *Member, IEEE*, Wanlei Zhou *Senior Member, IEEE*

Abstract—A critical challenge in machine learning is the vulnerability of learning models in defending attacks from malicious adversaries. In this research, we propose game theoretical learning between a variational adversary and a Convolutional Neural Network (CNN), participating in a variable-sum two-player sequential Stackelberg game. Our adversary manipulates the input data distribution to make the CNN misclassify the manipulated data. Our ideal adversarial manipulation is a minimum change to the data which yet is large enough to mislead the CNNs. We propose an optimization procedure to find optimal adversarial manipulations by solving for the Nash equilibrium of the Stackelberg game. Specifically, the adversary's payoff function depends on the data manipulation which is determined by a Variational Autoencoder, while the CNN classifier's payoff functions are evaluated by misclassification errors. The optimization of our adversarial manipulations is defined by Alternating Least Squares and Simulated Annealing. Experimental results demonstrate that our game-theoretic manipulations are able to mislead CNNs that are well trained on the original data as well as on data generated by other models. We then let the CNNs to incorporate our manipulated data which leads to secure classifiers that are empirically the most robust in defending various types of adversarial attacks.

Index Terms—Adversarial learning, Variational Autoencoders, Convolutional Neural Networks, Game theory, Nash equilibrium

1 Introduction

A significant robustness gap exists between machine perception and human perception in machine learning. A critical challenge in machine learning is the vulnerability of learning models to security attacks from malicious adversaries. Even innocuous perturbations to the training data can be used to manipulate the behaviour of the learning model in unintended and malicious ways.

Adversarial examples are hard to detect because learning models trained on limited data are required to produce expected output for every possible input. They are created by adversarial algorithms analyzing the training process of learning models under attack. The optimal attack policy is formulated as solving for (often nonlinear and non-convex) optimization problems. Adversarial examples are able to successfully mislead learning models as long as adversary's attack is planned, because the learning model cannot react to new samples after its completed training. Szegedy *et al.* [1] observed that neural network layers do not disentangle basis distributions from semantic information. Goodfellow *et al.* [2] proposed that deep neural networks learn discontinuous input-output mappings so that imperceptible perturbations to network's input increase deep networks prediction error. Such imperceptible perturbations are called adversarial examples. Adversarial examples are analysed as a property of high-dimensional dot products. To get adversarial perturbation, cost function training a deep network is linearized around current parameter settings. This method is called Fast Gradient Sign Method (FGSM). FGSM generates adversarial perturbations that are unperceivable to human eyes but will highly influence the prediction results of a deep neural network. For example, an adversarial perturbation added to the image of a "panda" misleads the GoogLeNet model into a "gibbon" [2].

To make the learned models more robust, adversarial algorithms should incorporate adversary into the training process of the learning models [3]. Game theory provides a framework to study interactions between learning model (or learner for short) and intelligent adversary (or adversary for short) in terms of evolving strategies between learner and adversary. In game theoretical adversarial learning, adversarial examples are generated by designing learning models under various attack scenarios assumed in adversary's strategy space.

In this paper, we propose adversarial manipulations created by a variational adversary operation in a generative strategy space. A Stackelberg game is a type of game theoretic models where the game is played in sequence (i.e., one is a leader while the other player follows). In this research, we assume the adversary is the leader and the CNN follows by retraining the classifier. While participating in a Stackelberg game with our variational adversary, the classifier is assumed to behave like a blackbox model. The adversarial cost function is then optimized in a stochastic optimization manner by solving a variable-sum game. The adversarial manipulations are iteratively searched and optimized in the game until the adversarial payoff function starts decreasing for targeted class labels. A Nash equilibrium in our game is a stochastic optimum where both the adversary and the classifier reach their optimal payoff function values. At Nash equilibrium, the adversarial examples are able to attack the targeted classifier models with both considerations of high misclassification error and low attack cost.

- *Aneesh S Chivukula, Xinghao Yang and Wei Liu are with the Advanced Analytics Institute, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia. E-mail: Aneesh.Chivukula@uts.edu.au, Xinghao.Yang@student.uts.edu.au, Wei.Liu@uts.edu.au. Corresponding Author: Wei Liu*
- *Tianqing Zhu and Wanlei Zhou are with the Centre for Cyber Security and Privacy, Faculty of Engineering and Information Technology, University of Technology Sydney, Australia. E-mail: Tianqing.Zhu@uts.edu.au, Wanlei.Zhou@uts.edu.au.*

Manuscript received X X, X; revised X X, X.

Our major contribution of this paper are the following:

- We formulate new attack strategies and adversarial algorithms in a game theoretical model with variational adversaries. Our attack strategies are able to mislead Convolutional Neural Networks (CNNs) in both two-label and multi-label image recognition settings.
- We propose optimization algorithms that use the Simulated Annealing (SA) and Alternating Least Squares (ALS) algorithms to search for Nash equilibrium in Stackelberg games. From the Nash equilibrium, we design a secure CNN classifier that is robust to adversarial attacks.
- We compare the proposed secure CNN classifier against other deep learning baseline methods, such as Generative Adversarial Networks (GANs) and other game theoretical adversarial learning models. Our experiments demonstrate that our model is the most effective to defend various types of adversarial attacks.

The paper starts with related work in Section 2. Theoretical formulation of game framework is in Section 3. The pseudocode for game model and its experimental validation are presented in Section 4 and Section 5, respectively. The paper ends in Section 6 with a summary of conclusion and future work.

2 Related Work

In this section we compare the proposed model with the state-of-art deep generative models and game theoretical adversarial learning models. Depending on adversary's knowledge of targeted model, attack scenarios in adversarial learning algorithms are categorized into blackbox attacks and whitebox attacks [3]. Typically, blackbox attacks query targeted model's output labels while whitebox attacks compute derivative of targeted model's loss function. We use derivative free optimization to introduce game theoretical objective functions into learning process of querying targeted model in a blackbox attack.

2.1 Stochastic Games in Predictive modelling

The interaction between an adversary and the classifier has been modelled as a Stackelberg Game. Here adversary's role is not that of a static data generator but an intelligent agent making deliberate data manipulations to evade classifiers. Re-learning classifier weights is a weak solution since evasion attacks are generated at cheaper and faster rate than re-learning.

Li *et al.* [4] proposed a feature cross-substitution attack to demonstrate objective-driven adversaries exploiting such limitations of feature reduction in adversarial settings. Adversary is able to query classifier according to a fixed query budget and a fixed cost budget. An adversarial evasion model with a sparse regularizer is then presented. Constructing the classifier on feature equivalence classes rather than feature space is proposed as a solution to improve classifier resilience.

Bianchi *et al.* [5] presented repeated games for random prediction problems. The problem of sequential prediction is modelled in the framework of Nash equilibrium found in normal form games. Specific minmax theorems are discussed to analyze two-player zero-sum games. A mistake bounds framework is provided to analyze game theoretical learning algorithms. Stochastic two-player zero-sum games incorporating multiple adversaries were analyzed by Ummels [6].

Zhou *et al.* [7] surveyed two-player and multiple player Stackelberg games in adversarial learning algorithms and cyber security applications. The interaction between adversary and classifier is modelled as one or more of simultaneous games, sequential games where adversary can be either a leader or a follower in the game. Alpcan *et al.* [8] presented large-scale strategic games and reduced games computation consumption and information limitation on Nash equilibrium solutions. Oliehoek *et al.* [9] proposed an deep generative adversary with resource-bounded best responses and Nash equilibrium on synthetic data. The generative adversary has a generator network and a discriminator network in a supervised learning problem and operated on discrete data. Papernot *et al.* [10] provided a threat model summarizing various attack scenarios in adversarial learning algorithms. The threats to machine learning models are adversarial manipulations, which are generated during both training process and inference process. Papernot *et al.* [10] also proposed no free lunch theorem and probably approximately correct model for adversarial learning. Moreover, Yang *et al.* [11] analyze the Nash equilibrium of a differential dynamical system modelling the Advanced persistent threat (APT) cyberattack scenario.

In this research, we propose new non-convex best responses in every play of the prediction game solving for the adversarial manipulations. Our bilevel stochastic optimization problem in the prediction game is formulated as a repeated sequential variable-sum two-player Stackelberg game. The optimization problem is solved by an Alternating Least Squares (ALS) search procedure that continuously attacks retrained classifier with adversarial manipulations optimized until Nash equilibrium. The ALS procedure evaluates candidate adversarial manipulations generated by a Simulated Annealing (SA) procedure for an increase in adversarial payoff function over the targeted class labels. Therefore, the adversarial data generated in the Stackelberg game simulates continuous interactions rather than one-time interactions with the learning processes of the classifier.

2.2 Adversarial Examples for Misleading Classifiers

In trying to interpret the solutions in deep neural networks, Szegedy *et al.* [1] introduced adversarial examples as discontinuities in input-output mappings learnt by deep neural networks. Goodfellow *et al.* [2] proposed a training regime called Fast Gradient Sign Method (FGSM) for generating adversarial examples, which can be computed efficiently using backpropagation. Papernot *et al.* [12] introduced a blackbox attack strategy where adversarial examples are generated without knowledge of target deep neural networks internal parameters and inputs. Baluja *et al.* [13] proposed a targeted attack where feed-forward neural networks called Adversarial Transformation Networks (ATNs) are trained to generate adversarial examples. ATNs generate adversarial examples that minimally modify classifier's outputs given original input. By contrast, Moosav *et al.* [14] constructed an untargeted attack technique, i.e., DeepFool, which is optimized by distance metrics between adversarial examples and normal examples. Biggio *et al.* [3] surveyed adversarial examples in pattern classifiers about deep neural networks applications in computer vision and cyber security. Adversarial attacks at classification's training time and testing time are called poisoning attacks and evasion attacks, respectively.

In our research, we generate adversarial examples to form a poisoning attack on the classification training data. The adversarial examples are generated by the adversarial manipulations learnt during our game theoretical attacks on the training process of

the learner. In our attack scenarios, no prior knowledge about the learning model is assumed. Our adversary knows neither the learning model's training process nor the learning model's best response strategies across the Stackelberg game's plays. Our adversary does a targeted attack to manipulate multiple positive labels into a single negative label. The attack strength of our adversarial manipulations is defined in terms of search randomization parameters in ALS and SA. The scalar optima in SA are used to generate the vector optima in ALS. The local optima in ALS converge onto the non-convex stochastic optima solving the Stackelberg game to output optimal adversarial manipulations. The optimal adversarial manipulations are able to encode the adversarial data in terms of the multivariate statistical parameters of a Gaussian mixture model produced in multi-label datasets.

2.3 Generative Adversarial Networks for Adversarial Learning

Generative Adversarial Networks (GANs) [15] estimate data likelihood with a two-player adversarial framework, which contains a generator network G and a discriminator network D. IWGAN [16] improves GAN with regularization but ignores the correlations between generated examples. InfoGAN [17] uses an information-regularized generator to disentangle interpretable representations from generated data. The distribution of adversarial perturbations have been modelled with AdvGAN [18] in whitebox attacks as well as blackbox attacks. Adversarial examples have been defined for deep generative models [19]. Importantly, we note that the fundamental difference between our research and the objective of generative networks is deceiving the classifier rather than mimicking the original data [20, 16]. We solve a supervised learning problem while deep generative models generally solve an unsupervised learning problem.

A taxonomy of adversarial attack scenarios in deep learning is provided by Gilmer *et al.* [21] and Biggio *et al.* [3]. Besides, another thread of research on adversarial autoencoders [22, 3] imposes a prior distribution on the output of an encoder network learning training data, where autoencoder discriminatively predicts whether a sample comes from its latent space or from prior distribution determined by the user. By contrast, our game theoretical optimization problem is independent from a particular training data distribution and classification model. The objective of our game formulation is not to improve classification accuracy by augmenting the original data training autoencoders.

Our attack scenario with Stackelberg games proposes new adversarial payoff functions. We represent feature space for adversarial manipulations in terms of adversarial cost functions, stochastic operators and game strategies in a simulated annealing algorithm.

2.4 Game theoretical adversarial learning

Dalvi *et al.* [23] analyzed classifier performance by viewing classification as a game with the classifier adapting to an adversary, aiming to make the classifier produce false negatives. Lowd *et al.* [24] introduced adversarial algorithms to learn a linear classifier's decision boundary. An ACRE learning framework is used to determine whether an adversary can efficiently learn enough about defeating a classifier by minimizing a linear adversarial cost function. Biggio *et al.* [25] defined poisoning attacks against Support Vector Machines (SVMs) by injecting adversarial examples into training data. A gradient ascent procedure computes adversarial examples as local maxima of SVM's non-convex error surface. Bruckner *et al.* [26] proposed prediction games to model interaction between a learner building predictive models and a data generator controlling data generation process. Kantarcioglu *et al.* [27] designed a subgame-perfect Nash equilibrium, which optimizes attribute selection with cost functions in an adversarial classification Stackelberg Game. Liu *et al.* [28] modelled competing behaviour between a rational adversary and a blackbox data miner as a sequential Stackelberg game where the payoff for each player is designed as a regularized loss function. Such a game is repeated until the adversarys payoff does not increase or the maximum number of iterations is reached.

Moreover, Wang *et al.* [29] assume that adversary changes any feature of the classifier at will and pays a cost proportional to size of the feature subset that has been changed. Such an attack on classifier is called sparse feature attack. The minmax optimization problem is formulated as a non-zero sum game. Chivukula *et al.* [30] enhanced [28] proposals for deep learning models while Yin *et al.* [31] extended them for sparse attack scenarios. Zhou *et al.* [32] explored a nested game framework, where adversarial strategy is chosen according to a probability of making prediction about classifier's decision boundary in a single leader multiple followers game.

Different from the above, in this research we define adversarial cost functions on a strategy space encoding original data distribution. Our adversarial payoff functions are optimized by a simulated annealing algorithm randomizing step changes in adversary's strategy spaces. Randomization in our adversarial attack strategies is defined by the latent space reconstructing original data distribution with a Variational Autoencoder (VAE). The proposed (variational nonlinear non-convex) adversarial cost function leads to a better regularization of the adversarial payoff function in our Stackelberg game.

2.5 Nash equilibria and Stackelberg Strategies

A Stackelberg game is a leader-initiated game, where adversarial strategies are modeled and solved for the solution rationale and decision-making problem defining the Nash equilibria. The solution space for Nash equilibria are expressed in terms of the necessary and sufficient conditions for game players' convergence criteria [33].

The optimization of such game theoretical payoff functions presents a complex problem in optimization theory. Such problems are often modelled as decision problems in noncooperative differential games [34]. In experiments on Stackelberg Strategies, we demonstrate the statistical significance of our optimal adversarial manipulations on the targeted multi-label classifiers decision boundaries. The decision boundaries are learnt from labelled image databases, datasets produced by existing deep generative models and datasets produced by existing game theoretical models for adversarial learning.

We empirically evaluate our adversarial learning algorithms in stochastic optimization settings. We generate adversarial manipulations at Nash equilibria that is found in the Stackelberg game. Our adversary's strategy space is determined by variational parameters learnt from the input data distribution. The optimal adversarial manipulations found by our adversarial algorithm define a deep generative model for creating the adversarial data in classifier's input data space.

3 GAME FORMULATION

In this section, we formulate a sequential variable-sum two-player Stackelberg game to mislead classifiers. Given training data distribution, our adversarial attack scenario seeks to find data manipulations that generate changed data distributions misleading the classifier in the game theoretical model. Effective attack scenario is measured in terms of a classifier's misclassification error and an adversary's manipulation cost. Manipulation cost measures adversary's expenses to generate data manipulations in each game iteration. After multiple game iterations, these adversarial manipulations are optimized to find a Nash equilibrium between the converged adversary and classifier.

We formulate the adversarial payoff function on a convex strategy space determined by a Variational Autoencoder (VAE) model. The VAE model encodes training data and validation data into unsupervised mean and variance codecs. These codecs represent multivariate Gaussian distributions that are derived from data distributions in a latent space. We assume this latent space is the strategy space of our game. Please note that the VAE is part of our modelling and is not part of the classification model that will be under attack. At Nash equilibrium, we produce an output which is a vector of adversarial manipulations corresponding to VAE's encodings of training data. We assume a Convolutional Neural Network (CNN) to be the classifier of multi-label image databases, and a Simulated Annealing (SA) algorithm is utilized in doing stochastic search and optimization. To simulate a blackbox attack, we assume that the adversary has no prior knowledge of the classifier's training process. CNN is adversarially retrained according to an alternating least squares (ALS) algorithm until the game convergence to the Nash equilibrium. In every game iteration, SA computes convex adversarial data manipulations as local optima to ALS, which in-turn finds local optima to non-convex game theoretical model for characterizing our attack scenario.

In our Stackelberg game, we model the adversary acting as a leader player (L) attacking targeted classifier model. The classifier responds to adversarial manipulation by acting as a follower player (F), defending its classification performance by adversarial retraining. The attack-defence game plays between L and F proceeds according to a sequence of increasing payoff function values. Such a game is called a sequential Stackelberg game. It can converge to Nash equilibrium with optimal payoff function values for both players. The optimal payoff brings an optimal data manipulation for the adversary.

All the potential manipulations/moves of L are assumed to be over a strategy space A^M, A^Σ, where M and Σ are mean and variance parameter spaces determined by a VAE model. The moves of F are assumed to be over classifier's parameter space W, which is learnt on training data distribution. During each game iteration, the move of L and F is determined by their real-valued payoff functions $J_L \in R$ and $J_F \in R$, respectively. To ease understanding, the variables notation used in our problems and algorithms are summarized in Table 1.

In every game play, the value of J_L and J_F depends on candidate adversarial manipulations $\alpha^\mu \in A^M, \alpha^\sigma \in A^\Sigma$ and candidate classifier weights $w \in W$. Adversary's best move finds a best strategy $\alpha^\mu_*, \alpha^\sigma_*$ with best payoff J_L as expressed in Eq. (1).

$$\alpha^\mu_*, \alpha^\sigma_* = argmax_{(\alpha^\mu \in A^M, \alpha^\sigma \in A^\Sigma)} J_L((\alpha^\mu, \alpha^\sigma), w) \tag{1}$$

In response to adversarial manipulation, the best move of retrained classifier then converges onto a robust $w^* \in W$ with best payoff J_F as expressed in Eq. (2).

$$w^* = argmax_{w \in W} J_F((\alpha^\mu, \alpha^\sigma), w) \tag{2}$$

Substituting Eq. (2) in Eq. (1) we have Eq. (3) in terms of the adversarial payoff function $J_L((\alpha^\mu, \alpha^\sigma), w)$ and classification payoff function $J_F((\alpha^\mu, \alpha^\sigma), w)$.

$$\begin{aligned}(\alpha^\mu_*, \alpha^\sigma_*, w^*) = & argmax_{(\alpha^\mu \in A^M, \alpha^\sigma \in A^\Sigma)} J_L\big((\alpha^\mu, \alpha^\sigma), \\ & argmax_{w \in W} J_F((\alpha^\mu, \alpha^\sigma), w) \big)\end{aligned} \tag{3}$$

Eq. (3) defines the Nash equilibrium in terms of final adversarial manipulations $\alpha^\mu_*, \alpha^\sigma_*$ and final classifier weights w^* that will be found after all the plays of the Stackelberg game.

In our blackbox attack scenario, we assume CNN misclassifying a set of positive labels Pos into another set of negative labels Neg. For $pos \in Pos$ and $neg \in Neg$, we map every label pair (pos, neg) to a sequential two-player Stackelberg game, i.e., solving Eq. (3) for multi-label adversarial manipulations $\mathbb{A}^\mu_*[pos, neg] = \alpha^\mu_*, \mathbb{A}^\sigma_*[pos, neg] = \alpha^\sigma_*$.

To solve Eq. (3), we assume that the movements between adversary and classifier interact in a sequence of game plays defined by a variable-sum Stackelberg game in Eq. (4)

$$J_L + J_F = \Phi + \lambda \times cost_L(\alpha^\mu, \alpha^\sigma) + cost_F(w) \tag{4}$$

where $\Phi \in R$ is a proportionality constant to increase adversary's payoff at the expense of classifier's payoff J_F. Our game is a variable-sum Stackelberg game because the R.H.S of Eq. (4) is a variable number depending on values of $cost_L$ and $cost_F$ computed in every game iteration.

$cost_L(\alpha^\mu, \alpha^\sigma)$ is the adversarial cost function, which typically has a closed form expression for game theoretical optimization and depends on application-specific adversarial data distributions. $\lambda \in R$ is a constant parameter to balance the weight between $cost_L$ and $cost_F$.

$cost_F(w)$ is the classifier's cost for doing adversarial retraining across multiple game plays. Typical $cost_F(w)$ has no closed form expression for game theoretical optimization. In our research, $cost_F(w)$ relies on CNN's retraining performance and optimization strategies, such as, retraining runtime, number of adversaries, player's attack and defence budgets, training data size and classifier weights initialization.

The interplay between adversary's strategy spaces and classifier's decision boundaries empirically determines the effect of $cost_L$ on $cost_F$. In blackbox attack scenario, we assume that the adversary's cost function $cost_L$ is independent from the classifier weights w. Similarly, the classifier's cost function $cost_F$ is independent from the adversarial manipulations $\alpha^\mu, \alpha^\sigma$.

Substituting J_F from Eq. (4) into Eq. (3) defines bilevel non-convex objective function for variational Stackelberg game:

$$\begin{aligned}(\alpha^\mu_*, \alpha^\sigma_*, w^*) = & argmax_{(\alpha^\mu \in A^M, \alpha^\sigma \in A^\Sigma)} J_L\big((\alpha^\mu, \alpha^\sigma), \\ & argmax_{w \in W}(\Phi + \lambda \times cost_L(\alpha^\mu, \alpha^\sigma) \\ & + cost_F(w) - J_L((\alpha^\mu, \alpha^\sigma), w))\big)\end{aligned} \tag{5}$$

Eq. (5) characterizes our blackbox attack scenario. The objective function is computed in terms of the adversarial payoff function J_L, which determines the candidate adversarial manipulations $\alpha^\mu, \alpha^\sigma$. J_L converges to Nash equilibrium in a two-player multi-label sequential Stackelberg variable-sum game. Nash equilibrium

TABLE 1: Table of Notation

Variable Name	Variable Description	Variable Name	Variable Description	Variable Name	Variable Description
L	Adversary as Leader in Stackelberg Game	$\mathbb{A}_*^\mu$	Multi-label adversarial manipulations on data mean parameters	$cost_F$	F's classification cost function
F	Classifier as Follower in Stackelberg Game	$\mathbb{A}_*^\sigma$	Multi-label adversarial manipulations on data variance parameters	α^μ	Adversarial manipulation over means in L's strategy space
M	training data mean parameter space	μ_{pos}, σ_{pos}	pos training data's mean and variance parameters	α_*^μ	L's best move for manipulating data mean parameters
Σ	training data variance parameter space	μ_{neg}, σ_{neg}	neg training data's mean and variance parameters	α_*^σ	L's best move for manipulating data variance parameters
A^M, A^Σ	Leader L's strategy space	δ_μ	Maximum possible difference between mean parameters of negative and positive training data	α^σ	Adversarial manipulation over variances in L's strategy space
W	F's training parameter space	δ_σ	Maximum possible difference between variance parameters of negative and positive training data	α_{next}	Candidate adversarial manipulation generated in SA iteration
$payoff_{next}$	L's payoff function value when data is manipulated for α_{next}	$payoff_{curr}$	L's payoff function value in current iteration	$\alpha_{initial}$	Initial adversarial manipulated in SA
J_L	L's adversarial payoff function	$error_{curr}$	F's misclassification error value in current iteration	$\alpha_{optimal}$	Optimal adversarial manipulated returned by SA
$cost_L$	L's adversarial cost function	$payoff_{best}$	L's best move payoff function value	α_{best}	Best adversarial manipulation element found by SA across mean and variance parameters
J_F	F's classification payoff function	$error_{best}$	F's best move misclassification error value	$error_{next}$	F's misclassification error value when data is manipulated for α_{next}

outputs game theoretical adversarial manipulations $(\alpha_*^\mu, \alpha_*^\sigma)$ and the corresponding secure classifier weights w^*.

To solve Eq. (5), we design adversarial payoff function J_L in Eq. (6) with $cost_L(\alpha^\mu, \alpha^\sigma) = (\|\alpha^\mu\|_F + \|\alpha^\sigma\|_F)$.

$$J_L((\alpha^\mu, \alpha^\sigma), w) = error_F(w) - \lambda * cost_L(\alpha^\mu, \alpha^\sigma) \quad (6)$$

The strategy space A^M, A^Σ for $(\alpha^\mu, \alpha^\sigma)$ is a randomized convex data space determined by a VAE, i.e., a multivariate Gaussian mixture model. J_L in Eq. (6) is optimized by a SA algorithm that does stochastic search and local optimization for $\alpha_*^\mu, \alpha_*^\sigma$ in every game play. $error_F(w)$ denotes CNN's misclassification performance for positive labels $pos \in Pos$ to be manipulated into negative labels $neg \in Neg$.

During game model training evaluations, we define $error_F(w)$ of Eq. (6) in terms of true positive rate (of pos when CNN misclassifies pos label data as neg label data): $error_F(w) = error_{pos}(w) = (1 - tpr_{pos}(w))$ and call it attack performance. Here $tpr_{pos}(w) = \frac{tp_{pos}(w)}{p_{pos}(w)}$.

Eq. (6) is sub-problem of Eq. (5) which is the objective function for game theoretical optimizations. Eq. (6) is solved repeatedly by the adversary until convergence of the game iterations. In every game iteration, the changes to attack parameters $(\alpha^\mu, \alpha^\sigma)$ adversarially influence the updates to classification parameters w. Such a game's convergence condition is called the Nash equilibrium. It is a balance of maximum increases to payoff functions values J_L and J_F solving Eq. (5) until convergence.

After game converges to optimum data manipulations we evaluate game model performance on testing data by defining the $error_F(w)$ in terms of (percentage) classification f_1-score for Pos as $error_F(w) = error_{Pos}(w) = (1 - fscore_{Pos}(w))$ for all $Pos \times Neg$ and call it defence performance. Here $fscore_{Pos}(w) = 2 \times \frac{tp_{Pos}(w)}{tp_{Pos \cup Neg}(w) + fn_{Pos \cup Neg}(w) + fp_{Pos \cup Neg}(w)}$. To simplify the experimental setup we have assumed $Pos = pos$ and $Neg = neg$ for evaluating defence performances in Section 5. However the CNN classifier is trained on all possible class labels $(pos, neg) \in (Pos \times Neg)$.

Across the Stackelberg game iterations, we design the SA to optimize the $cost_L$ such that the leader L gains in changes to its payoff J_L is in proportion to the payoff changes lost by the follower F in changes to its payoff J_F. In Eq. (4), we also allow the follower F to assume an independent blackbox cost $cost_F$ of retraining to engage the leader L in a Stackelberg game. In Eq. (6), the pure profit for the adversary (acting as leader L) is determined by $cost_L$. While trying to minimize $cost_L$ the adversary's target is to maximize the misclassification error $error_F$ of a CNN (acting as follower F). $error_F$ inturn depends on the CNN's architecture and loss function which we assume are unknown to the adversary according to a blackbox attack scenario. By including $error_F$ in Eq. (6), we only seek to model the adversarial manipulations to training data distributions. We define Eq. (6) with reference to misclassification error $error_F$ but not misclassification cost $cost_F$ because we assume that the adversary targets the CNN by manipulating its input data without knowing any more detail of its retraining procedure for participating in the game. By analyzing the convergence of the adversarial cost function $cost_L$ but not the classification cost function $cost_F$, we analyze the stochastic optima of a sequential Stackelberg game rather than a simultaneous Stackelberg game. We solve the sequential Stackelberg game of Eq. (4) by optimizing the adversary L's cost $cost_L$ for leading the sequential game.

In Eq. (6), $error_F$ can also be interpreted as a regularization term on $cost_L$ to be optimized in Eq. (4). Here λ controls the relative importance of adversarial cost function $cost_L$ compared to misclassification error $error_F(w)$ for creating adversarial manipulations $\alpha_*^\mu, \alpha_*^\sigma$ in each game iteration. The tradeoff between decreasing cost $cost_L$ and increasing error $error_F(w)$, with respect to optimization parameters solving for our blackbox attack scenario, plays out across all of the search iterations described in our game theoretical algorithms of Section . L's strategy space in Algorithm 1 and Algorithm 2 is defined on the original data distributions and encoded data distributions leading to global optima solving for the Nash equilibrium of the Stackelberg game. L's strategy space in Algorithm 3 is determined by the SA parameters solving for the local optima in step and direction of each

adversarial manipulation. When we consider all the adversarial manipulations learnt by SA parameters in a variational model, we solve for a stochastic optimization problem that is nonlinear and non-convex in the terms of the objective function in Eq. (4).

Along with SA parameter settings described above, VAE's encoder function Enc and decoder function Dec determine our game theoretical attack scenarios. Given training data X_{train}, we generate adversarial data X_{gen} via:

$$X_{gen} = Dec(\mu_{pos} + \alpha^{\mu}, \sigma_{pos} + \alpha^{\sigma}) \cup X_{train}[neg], \qquad (7)$$

where $\mu_{pos}, \sigma_{pos} = Enc(X_{train}[pos])$. The Eq. 7 first encodes positive data distribution $X_{train}[pos]$ into multivariate Gaussian distribution's parameters $(\mu_{pos}, \sigma_{pos})$. Then it adds adversarial manipulations $(\alpha^{\mu}, \alpha^{\sigma})$ to $(\mu_{pos}, \sigma_{pos})$. After that, the encoded adversarial data is decoded by Dec into original data space. The distribution of negative data $X_{train}[neg]$ is unchanged since we target pos class performance of CNN classifier.

3.1 Overall Structure of Our Model

Figure 1 is a flow chart of our adversarial learning process that accounts for the presence of a variational adversary in supervised learning. The final outcome of our adversarial learning is a CNN classification model CNN_{secure} (henceforth shortened as CNN_s) that is robust to the adversarial attacks.

We generate the adversarial data in a two-player Stackelberg game between the adversary and the classifier. The adversary creates a variational model by searching for adversarial manipulations on encoded training data. Every statistical parameter of the encoded training data is searched according to a Simulated Annealing (SA) procedure. The aggregation of adversarial manipulations to all statistical parameters in the encoded training data is optimized according to an Alternating Least Squares (ALS) procedure. The ALS optimization is invoked at each time when the adversary generates adversarial data X_{gen} in the Stackelberg game. X_{gen} acts as a validation data for the classifier under attack. For every X_{gen}, the classifier re-optimizes its training weights to update itself.

The result of such a game-theoretic interaction between the learner's and classifier's best moves is quantified by the adversary's payoff $payoff_{best}$. The adversary engages the classifier in the Stackelberg game as long as the $payoff_{best}$ increases. A decrease of $payoff_{best}$ indicates that Nash equilibrium exit condition has been reached in the Stackelberg game. At the end of the game, adversary has optimal adversarial manipulations from the most recent X_{gen}. Such manipulations are applied on the training data to obtain attacked training data. Then the classifier's learning process adds the attacked data into the original training data so that the CNN_s can be optimally retrained by our adversarial attacks. While the CNN classifier is trained in the original data space, the adversary generates data manipulations in the encoded data space. A variational representation of the encoded data space allows the adversary to propose a generative model for the adversarial manipulations.

3.2 The differences between our method and GANs

Although both GAN and our method are based on the framework of game theory and both of them are seeking for the Nash equilibrium, they have some great differences. Firstly, GAN's objective is to learn a generative model that mimic the original distribution of data, while our method learns the optimal adversarial manipulations $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ that are not the original true distribution of the data but are *manipulations* to the original distribution. Secondly, the GAN at its optima is to best distinguish synthetic images from true ones that belong to the same class label (i.e., a generative learning problem), rather than to classify labels of synthetic images at the highest accuracy (i.e., a label classification problem). From this perspective, the GAN may not be expected to perform well when an image is deliberately and subtly changed to mislead the classifier to misclassify it to a targeted wrong label. Thirdly, in our attack scenario, different proposals on adversarial payoff functions and adversarial cost functions in Eq. (6) lead to different Nash equilibria for Eq. (5) and corresponding adversarial manipulations in Eq. (3). In contrast, a GAN always tries to converge the synthetic data to the original training data distributions.

3.3 Variational Game and Adversarial Examples Illustration

To demonstrate examples of successful attacks, Figure 2 shows adversarially manipulated samples generated by our model. For the two images in each subfigure of Figure 2, the left one is the original image while the right is the manipulated image. The original images shown for specific target classes are from the MNIST database and the VGGFace2 database. We evaluate adversary's payoff in Eq. (6) over data distributions of many such manipulated and misclassified images.

Specifically, Figure 2(a) to Figure 2(e) show adversarially manipulated MNIST images that have been misclassified to the class label 8. In Figure 2(a) and Figure 2(d), handwritten digits 2 and 6 have been manipulated by deleting pixels to smoothen sharp edges. In Figure 2(b) and Figure 2(c), handwritten digits 3 and 5 have been manipulated by adding and deleting pixels to change orientation. In Figure 2(e), handwritten digit 9 has been manipulated by adding pixels targeting particular aspects of the image geometry that are similar between images of 9 and 8.

Figure 2(f) to Figure 2(j) show adversarially manipulated VGGFace2 images that have been misclassified as belonging to the class label "Jackie Chan". In Figure 2(f) and Figure 2(h) images of "Andy Lau" and "Zhang Jingchu" have been manipulated by blurring parts of the facial features while leaving the remaining background unchanged. In Figure 2(g) and Figure 2(i) images of "Ajay Devgan" and "Aishwarya Rai Bachchan" have been manipulated by changing the shapes of sunglasses and hand covering the targeted classes facial features. In Figure 2(j), image of "Jada Pinkett Smith" has been manipulated by decreasing the distinction between the colours of the image foreground and the image background.

4 Variational Stackelberg Game Method

The pseudocode for two-player game which outputs multi-label adversarial manipulations $(\mathbb{A}^{\mu}_{*}, \mathbb{A}^{\sigma}_{*})$ is given in Algorithm 1. Each pair (pos, neg) of positive labels $pos \in Pos$ and negative labels $neg \in Neg$ executes a separate two-player game that misleads classifier into misclassifying pos data as neg data. The input to Algorithm 1 is labelled training data X_{train} consisting of grayscale images. A labelled testing data X_{test} is separately used to evaluate $(\mathbb{A}^{\mu}_{*}, \mathbb{A}^{\sigma}_{*})$ at the end of each game. Algorithm 1 also defines some input parameters to characterize the game theoretical adversary and find an optimal attack from the strategy space. Candidate adversarial manipulations $(\alpha^{\mu}, \alpha^{\sigma})$ search and optimization over

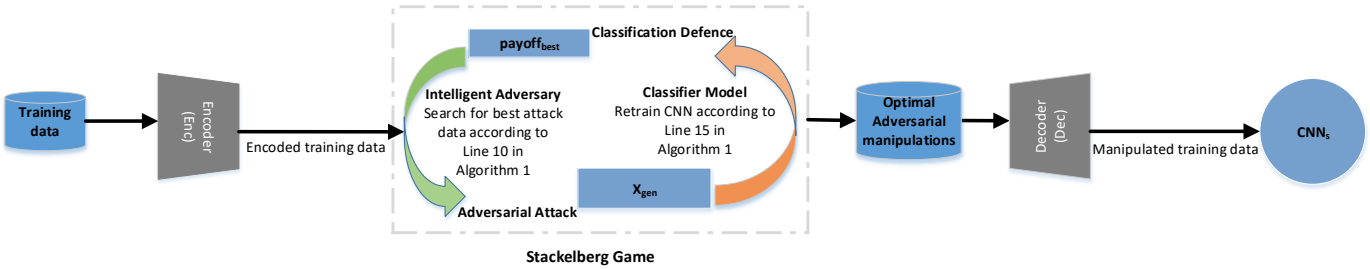


Fig. 1: A flowchart illustrating our overall game theoretical adversarial modelling. The data is encoded before entering the game and undertake simulated attacks. From the encoded data, a Stackelberg game discovers (from its Nash equilibrium) the optimal adversarial manipulations. The manipulations are then decoded and added to the original data to formulate a new training set for our CNN_s model.


(a) 2 misclassified by CNN as 8


(b) 3 misclassified by CNN as 8

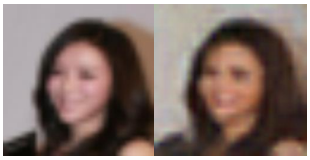

(c) 5 misclassified by CNN as 8

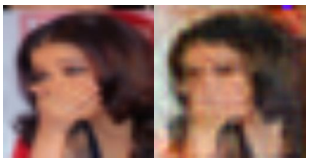

(d) 6 misclassified by CNN as 8


(e) 9 misclassified by CNN as 8


(f) Andy Lau misclassified by CNN as Jackie Chan


(g) Ajay Devgan misclassified by CNN as Jackie Chan


(h) Zhang Jingchu misclassified by CNN as Jackie Chan


(i) Aishwarya Rai Bachchan misclassified by CNN as Jackie Chan


(j) Jada Pinkett Smith misclassified by CNN as Jackie Chan

Fig. 2: Examples of transformed images found at Nash equilibrium in a Stackelberg game. For the two images in each subfigure, the left one is the original image and the right is the manipulated image.

the adversarial payoff function $J_L = payoff_{best}$ is described in the Alternating Least Squares algorithm of Algorithm 2 and Simulated Annealing algorithm of Algorithm 3.

Line 1 of Algorithm 1 trains a classifier model CNN on X_{train} to initialize three models $CNN_{original}$, $CNN_{manipulated}$ and CNN_{secure} (henceforth shortened as CNN_o, CNN_m and CNN_s, respectively) with same weights that provide a high classification performance on X_{test}. During the game, adversary repeatedly attacks CNN classifier with data manipulations $(\alpha_*^{\mu}, \alpha_*^{\sigma})$. As a response, CNN classifier is allowed retraining its weights w^* by using both X_{gen} and X_{train} on line 15. At the end of the game, adversary converges to data manipulations $(\mathbb{A}_*^{\mu}[pos, neg], \mathbb{A}_*^{\sigma}[pos, neg])$, which gives the highest value for adversarial payoff function $J_L = payoff_{best}$.

Then, CNN_o is evaluated on testing data X_{test} to output original performance $error_{original}$ (henceforth shortened as $error_o$) on line 20. On line 21, $CNN_{manipulated}$ is evaluted on manipulated testing data $X_{test-manip}$ as well as testing data X_{test} to output attack performance $error_{pos}(w) = error_{manipulated}$ (henceforth shortened as $error_m$). On line 22, CNN_m is retrained on manipulated training data $X_{train-manip}$ to output secure classifier CNN_s. On line 23, CNN_s is evaluated on manipulated testing data $X_{test-manip}$ and testing data X_{test} to output defence performance $error_{Pos}(w) = error_s$ (henceforth shortened as $error_s$). Notably, $error_o$, $error_m$, $error_s$ are calculated for each $(pos, neg) \in (Pos \times Neg)$.

Before the game begins, CNN_o's weights are learnt on training data X_{train} and CNN_m's weights are duplicate of CNN_o's weights. After the game ends, CNN_m weights and CNN_o weights remain unchanged. In the presence of an adversary, CNN_s weights are more robust than CNN_o weights. Classifier model is called CNN. CNN is initially trained on line 1, then it is adversarially trained on line 15 for every adversarial attack. CNN is discarded after game ends for each label pair (pos, neg).

The adversary's strategy space in Algorithm 2 is determined by an Encoder function Enc and a Decoder function Dec of a VAE model. The VAE encodes input data in terms of a mean vector μ and a variance vector σ to represent a multivariate Gaussian distributions in the latent space. In Algorithm 2, the size of latent space is given as attack scenario parameter s. On line 4 of Algorithm 1, Enc is applied on positive data $X_{train}[pos]$ and negative data $X_{train}[neg]$ to output positive mean vector μ_{pos} and negative mean vector μ_{neg}; positive variance vector σ_{pos} and negative variance vector σ_{neg}, respectively. These vectors are used on line 5 to derive vectors δ_{mu} and δ_{sigma} that measure amount of change in latent space to entirely convert encoded positive data $Enc(X_{train}[pos])$ into encoded negative data $Enc(X_{train}[neg])$.

The adversarial learning targets to search for small random

Algorithm 1: Variational Stackelberg Game Algorithm

Input: Training data X_{train}, Labelled testing data X_{test} (for final evaluations only), Training labels $(Pos \times Neg)$, Iteration error threshold's $maxerror\_game = 15$
Output: Adversarial manipulations in variational game $\mathbb{A}^{\mu}_{*}, \mathbb{A}^{\sigma}_{*}$, Original performance $error_o$, Attack performance $error_m$, Defence performance $error_s$

1 Train CNN on X_{train} to output model CNN_o. Initialize CNN_m, CNN_s with CNN_o's weights.
2 Train Variational Autoencoder VAE on X_{train} to get Encoder function Enc, Decoder function Dec.
3 **for** $(pos, neg) \in (Pos \times Neg)$ **do**
4 $\mu_{neg}, \sigma_{neg} = Enc(X_{train}[neg]), \mu_{pos}, \sigma_{pos} = Enc(X_{train}[pos])$
5 $\delta_{\mu} = mean(\mu_{neg}) - mean(\mu_{pos}), \delta_{\sigma} = mean(\sigma_{neg}) - mean(\sigma_{pos})$
6 Initialize zeros tensors α^{μ}_{*} and α^{σ}_{*} with same shape as $mean(\mu_{pos})$ and $mean(\sigma_{pos})$ respectively
7 exitgame = False, $payoff_{curr} = error_{curr} = -\infty, CNN = CNN_o$
8 **while** *exitgame* = *False* **do**
9 Generate adversarial data X_{gen} by substituting $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ in Eq. (7). Evaluate CNN on X_{gen} to find error $error_{curr}$. Substitute $error_{curr}$ in Eq. (6) to calculate $payoff_{curr}$ for $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$.
10 Update α^{μ}_{*} and α^{σ}_{*} from alternating least squares ALS in Algorithm 2
11 Generate adversarial data X_{gen} by substituting $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ in Eq. (7). Evaluate CNN on X_{gen} to find error $error_{best}$. Substitute $error_{best}$ in Eq. (6) to calculate $payoff_{best}$ for $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$.
12 **if** $payoff_{best} - payoff_{curr} > 0$ **then**
13 **if** $error_{best} > maxerror\_game$ **then**
14 exitgame = True, break *//Nash equilibrium achieved*
15 Re-optimize CNN weights on manipulated data and training data $X_{gen} \cup X_{train}$
16 $X_{train-manip} = X_{train-manip} \cup X_{gen}$
17 $\mathbb{A}^{\mu}_{*}[pos, neg], \mathbb{A}^{\sigma}_{*}[pos, neg] = \alpha^{\mu}_{*}, \alpha^{\sigma}_{*}$
18 **for** $(pos, neg) \in (Pos \times Neg)$ **do**
19 Simulate attacks by $\mathbb{A}^{\mu}_{*}[pos, neg], \mathbb{A}^{\sigma}_{*}[pos, neg]$ on $X_{test}[pos, neg]$ to output manipulated testing data $X_{test-manip}[pos, neg]$
20 Evaluate CNN_o on $X_{test}[pos, neg]$ to find original model error $error_o[pos, neg]$ for targeted label pos in attack scenario
21 Evaluate CNN_m on $X_{test-manip}[pos, neg] \cup X_{test}[pos, neg]$ to find manipulated model error $error_m[pos, neg]$
22 Train CNN_m on $X_{train-manip}[pos, neg]$ to output secure model CNN_s
23 Evaluate CNN_s on $X_{test-manip}[pos, neg] \cup X_{test}[pos, neg]$ to find secure model error $error_s[pos, neg]$ for targeted label pos
24 **return** $(\mathbb{A}^{\mu}_{*}, \mathbb{A}^{\sigma}_{*}, error_o, error_m, error_s)$

changes $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ to attack the encoded positive data $(\mu_{pos}, \sigma_{pos})$ on line 9 and line 11 such that CNN predicts a negative label neg for the manipulated positive data $(\mu_{pos} + \alpha^{\mu}_{*}, \sigma_{pos} + \alpha^{\sigma}_{*})$ after it is decoded into training data space. Since search for $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ is in VAE's latent space, the adversary's strategy space for attacker's payoff calculation in the two-player game is the VAE's latent space. However, the classifier's strategy space for defender's payoff calculation is the original data space, which is the same as reconstructed data space.

Line 9 evaluates current CNN on X_{gen} to find classification error $error_{curr}$. Then it computes adversarial payoff function value $payoff_{curr}$ by combining $error_{curr}$ with adversarial cost function, where the adversarial cost function is defined as the Forbenius norm of corresponding adversarial manipulation $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$. λ is a weight constant that controls the contribution of adversarial cost function when calculating the adversarial payoff function. In a given game execution and training data distribution, λ is set by adversary to increase payoff function by increasing classifier's error and decreasing adversary's cost across game plays.

On line 10, the Alternating Least Squares method (see Algorithm 2) updates candidate adversarial manipulations $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$. Line 11 updates classifier error $error_{best}$ and adversarial payoff $payoff_{best}$ corresponding to adversarial data X_{gen}. The looping condition on line 12 continues game plays as long as $payoff_{best}$ is greater than $payoff_{curr}$. All game plays end on line 17 when adversary converges to the optimal adversarial manipulation $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ for label pair (pos, neg).

4.1 Alternating Least Squares Algorithm

Algorithm 2 illustrates the Alternating Least Squares (ALS) search procedure, which optimizes the adversarial manipulations by changing the elements of vectors $\alpha^{\mu}_{best}, \alpha^{\sigma}_{best}$. By using the Simulated Annealing (SA) method (see Algorithm 3) on line 5

Algorithm 2: Alternating Least Squares (ALS)

Input: $\alpha^{\mu}_{*}, \alpha^{\sigma}_{*}, \delta_{\mu}, \delta_{\sigma}, payoff_{curr}, error_{curr}, \mu_{pos}, \sigma_{pos}, s$
Output: $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$

1 Initialization: Encoding space code size $s = 50$, $maxerror\_als = 10$, $exitsearch = False$
2 **while** $\neg$ *exitsearch* **do**
3 $payoff_{best}, error_{best} = payoff_{curr}, error_{curr}$
4 $\alpha^{\mu}_{best}, \alpha^{\sigma}_{best} = \alpha^{\mu}_{*}, \alpha^{\sigma}_{*}$
5 By fixing α^{σ}_{best}, update $\alpha^{\mu}_{best}, payoff_{best}, error_{best}$ from simulated annealing SA in Algorithm 3 for given δ_{μ} and μ_{pos}
6 By fixing α^{μ}_{best}, update α^{σ}_{best},$payoff_{best}, error_{best}$ from simulated annealing SA in Algorithm 3 where δ_{σ} replaces δ_{μ} and σ_{pos} replaces μ_{pos}
7 **if** $payoff_{best} - payoff_{curr} > 0$ **then**
8 **if** $error_{best} > maxerror\_als$ **then**
9 exitsearch = True, break
10 **else**
11 exitsearch = False
12 **else**
13 exitsearch = True
14 $payoff_{curr}, error_{curr} = payoff_{best}, error_{best}$
15 $\alpha^{\mu}_{*}, \alpha^{\sigma}_{*} = \alpha^{\mu}_{best}, \alpha^{\sigma}_{best}$
16 **return** $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$

and line 6, ALS can alternatively change each element of mean vector α^{μ}_{best} and variance vector α^{σ}_{best}, while fixing all the other elements. ALS optimizations are repeated until adversarial payoff function value $payoff_{best}$ stops increasing in comparison to current payoff function value $payoff_{curr}$. Starting with unchanged $(\alpha^{\mu}_{best}, \alpha^{\sigma}_{best})$ on line 4 of Algorithm 2, we empirically assume an attack scenario where the variance vector α^{σ}_{best} is changed only after mean vector α^{μ}_{best} has been changed in elemental sequence. The maximum number of elements to change is determined by VAE's code size s. After executing multiple search iterations, ALS finds optimally changed $(\alpha^{\mu}_{*}, \alpha^{\sigma}_{*})$ on line 16 of Algorithm 2. Thresholding $error_{best}$ with input parameter $maxerror\_als$ on

Algorithm 3: Simulated Annealing (SA)

Input: $\alpha_{best}, \delta_\mu, payoff_{curr}, error_{curr}, \mu_{pos}, s$
Output: $\alpha_{optimal}, payoff_{optimal}, error_{optimal}$
1 Initialization: Cost weighting constant $\lambda = 1$, Maximum number of steps $N = 10$, Lower bound step interval $l = 1$, Upper bound step interval $h = 5$, Step decrement $d = 0.5$ on h, Upper bound iteration count $maxiter = 100$, $maxerror\_sa = 5$, $\alpha_{optimal} = \alpha_{initial} = \alpha_{best}$
2 $payoff_{optimal} = error_{optimal} = -\infty$
3 **for** $i \in [0, s]$ **do**
4 $\alpha_{next} = \alpha_{initial}$
5 $\epsilon = \frac{\delta_\mu[i]}{N}$
6 exitoptimize = False, $iter = 0$,$jumped = False$
7 **while** $\neg$ *exitoptimize* $\wedge$ *iter* $< maxiter$ **do**
8 $manip_{next} \mathrel{+}= \epsilon \times random(l, h - d \times iter)$
9 $\alpha_{next}[i] \mathrel{+}= manip_{next}$
10 Generate adversarial data X_{gen} by substituting α_{next} in Eq. 7
11 Evaluate CNN on X_{gen} to find error $error_{next}$.
12 Substitute $error_{next}$ in Eq. 6 to calculate $payoff_{next}$
13 **if** $payoff_{next} - payoff_{curr} > 0$ **then**
14 $payoff_{curr} = payoff_{next}, error_{curr} = error_{next}$
15 **if** $error_{next} > maxerror\_sa$ **then**
16 exitoptimize = True, break
17 **else**
18 exitoptimize = False
19 **else**
20 **if** $error_{next} > error_{curr} \wedge jumped == False$ **then**
21 $jumped = True$,exitoptimize = False
22 $manip_{next} \mathrel{+}= \epsilon \times random(l, h)$
23 **else**
24 exitoptimize = True
25 $iter \mathrel{+}= 1$
26 $\alpha_{best} = \alpha_{initial}, \alpha_{best}[i] = \alpha_{next}[i]$
27 Generate adversarial data X_{gen} by substituting α_{best} in Eq. 7
28 Evaluate CNN on X_{gen} to find error $error_{best}$
29 Substitute $error_{best}$ in Eq. 6 to calculate $payoff_{best}$ for α_{best}
30 **if** $payoff_{best} > payoff_{optimal}$ **then**
31 $\alpha_{optimal} = \alpha_{best}$
32 $payoff_{optimal} = payoff_{best}, error_{optimal} = error_{best}$
33 **return** $\alpha_{optimal}, payoff_{optimal}, error_{optimal}$

line 8 ensures that the ALS search procedure exits as soon as least squares fit of $payoff_{best}$ is found by alternating SA optimizers, operating on the geometric surface of the adversarial payoff function J_L.

4.2 Simulated Annealing Algorithm

Algorithm 3 is the derivative-free stochastic optimization procedure for finding adversarial manipulations in terms of simulated annealing (SA) steps to i-th elements of vectors $(\alpha^{\mu}_{best}, \alpha^{\sigma}_{best})$. The step size in SA is set by ϵ on line 5. The bounds for ϵ depend on i-th element of δ_μ defined by Algorithm 1 in the latent space of the VAE. The magnitude of ϵ is determined by the parameter N dividing $\delta_\mu[i]$ into multiple increments. The direction of SA is set by the sign of $\delta_\mu[i]$ determining additive distance between negative data and positive data. The initial candidate for SA is $\alpha_{next}[i]$ where α_{next} has been initialized to candidate adversarial manipulation α_{best}. α_{best} is determined by line 4 of Algorithm 2.

For each $\alpha_{next}[i]$, candidate adversarial manipulations α_{next} are updated on line 9 where $manip_{next}$ is a random multiple of ϵ steps on line 8. The upper bound for selecting the random multiple of steps decreases with SA's iterations to reduce step size across SA iterations. After minimizing adversarial cost function by a greedy optimization strategy, we find a $manip_{next}$ that is close to 0 but is able to mislead CNN in $payoff_{next}$ calculation on line 12. Moreover, $payoff_{next}$ is expected to have low cost of optimization in adversary's strategy space (determined by VAE model) for large classification error in classifier's strategy space (which is the same as the original data space).

SA's convergence criteria are VAE's encoding space code size s, maximum number of SA steps N, adversarial cost function's weighting constant λ. The SA's optimization condition of increasing payoff $payoff_{next}$ is on line 13. The SA's jump condition of increasing error and exit condition of decreasing payoff as well as error are on line 20 and line 23, respectively. To find a successful adversarial manipulation, SA's optimization condition is expected to be frequently true while SA's jump condition is rarely true. On line 15, SA's optimization of $error_{next}$ is thresholded by parameter $maxerror\_sa$. Line 26 updates only the i-th element of α_{best} with i-th element of α_{next} found by SA. On line 28 and 29, $error_{best}$ and $payoff_{best}$ are calculated to obtain classifier's error and adversary's payoff for i-th adversarial manipulation. The sorting condition on line 30 finds $\alpha_{optimal}$ with highest adversarial payoff produced across all the s SA optimizers in stochastic optimization procedure of Algorithm 3.

5 Experiments

In this section, we validate the game theoretical adversary's attack scenarios in terms of CNN classifer models' performance validation. The attack scenarios are expressed in terms of stochastic optimization parameters in Algorithm 3. The number of times the adversary invokes Algorithm 3 is determined by the VAE's encoder codesize s. For each label pair (pos, neg), the rate of convergence of Algorithm 3 onto optimal payoff is determined by adversarial cost function's weighting constant λ. The step size of annealing operation in Algorithm 3 has upper bound N.

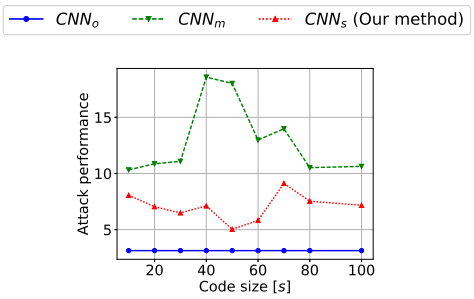
5.1 Classifier and Autoencoder Description

To create training data, testing data and validation data for game theoretical adversary, we conduct experiments on the handwritten images of MNIST database [35] and VGGFace2 database of human faces [36]. During the game, the adversary targets the targeted class label pos's true positive rate, which is also called the attack performance. After the game, adversarial data is used to measure CNN's f_1-score for varying pos label, which is defined as the defence performance. All CNNs and VAEs are implemented in Pytorch[1] platform. In each VAE, encoder has fully connected layers representing input data in terms of a multivariate Gaussian distribution with mean vector and variance vector of size s. The VAE's decoder architecture is a mirror of its encoder.

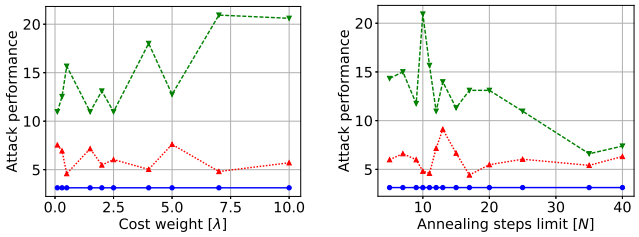
MNIST CNN and VAE: Following the loss function defined by Krizhevsky *et al.* [37], the CNN trained for MNIST has two convolution layers and two fully connected layers leading upto a crossentropy loss function with dropout regularization. The CNN has a learning rate of 0.01 and momentum of 0.5. It is trained for 25 epochs. Following the loss function defined by Kingma *et al.* [38], the MNIST VAE's encoder has two convolution layers. All the input images are in the size of 28×28 pixels. The VAE is trained for 50 epochs.

VGGFace2 CNN and VAE: Following the loss function defined by He *et al.* [39], the CNN trained for VGGFace2 is a pretrained model obtained from torchvision package[2]. The CNN has a learning rate of 0.01 and momentum of 0.9. It is trained for 100

[1] https://pytorch.org/docs/stable/index.html
[2] https://pytorch.org/docs/stable/_modules/torchvision/models/resnet.html

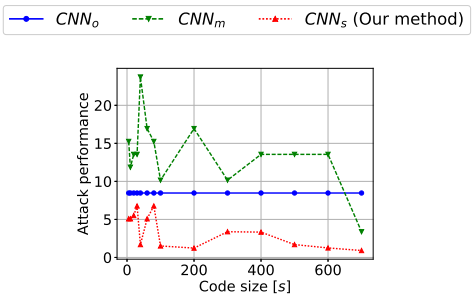


(a) Attack: varying s

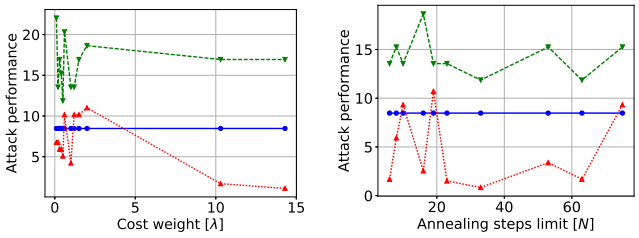


(b) Attack: varying λ (c) Attack: varying N

Fig. 3: MNIST Testing performance (error) with variations in attack parameters consisting of encoder code size s, adversarial cost weight λ and annealing steps limit N.



(a) Attack: varying s



(b) Attack: varying λ (c) Attack: varying N

Fig. 4: VGGFace2 Testing performance (error) with variations in attack parameters consisting of encoder code size s, adversarial cost weight λ and annealing steps limit N.

epochs. Following the loss function defined by Hou *et al.* [40], the VGGFace2 VAE's encoder has four convolution layers with batch normalization. All the input images are in the size of 32×32 pixels. The VAE is trained for 1000 epochs.

5.2 Attack Performance Validation

Figure 3 and Figure 4 show the variation in (percentage) attack performance for CNN_o, CNN_m, CNN_s on the y-axis when $pos = 5$ and $neg = 8$ in MNIST database and $pos = AndyLau$ and $neg = JackieChan$ in VGGFace2 database respectively. The x-axis in Figure 3(a) and Figure 4(a) varies s the code size of mean vector μ_{pos} and variance vector σ_{pos} encoding positive data. s is varied between 10 and 100 for MNIST database. s is varied between 5 and 700 for VGGFace2 database. The x-axis in Figure 3(b) and Figure 4(b) varies λ the weighting constant on adversarial payoff function. λ is varied between 0.1 and 10 for MNIST database. λ is varied between 0.1 and 15 for VGGFace2 database. The x-axis in Figure 3(c) and Figure 4(c) varies N the maximum number of steps taken by the adversary applying ALS of Algorithm 2 to optimize the step change to each element of the encoded data parameters μ_{pos} and σ_{pos}. N is varied between 5 and 40 for MNIST database. N is varied between 6 and 75 for VGGFace2 database.

5.3 Defence Performance Validation

While s determines the invocation times of SA in Algorithm 3, λ determines the change rate of $payoff_{best}$ crossing iterations in Algorithm 3. N sets the infinitesimal step size ϵ in Algorithm 3.

For each attack parameter value (and adversarial dataset) fixed on the x-axis, each data point triplets marked on y-axis. Figure 3 and Figure 4 show that the manipulated CNN (CNN_m) has higher error than the original CNN (CNN_o), and secure CNN (CNN_s) has lower error than manipulated CNN (CNN_m). Thus adversarial data has successfully misled CNN_o into misclassifying pos data as neg data, which is measured by true positive rate of pos in CNN's predictions. The exact change in error between CNN_m and CNN_o is determined by the game's iterations thresholds, i.e., $maxerror\_game$, $maxerror\_als$ and $maxerror\_sa$, which act as the exit condition on the game's convergence criteria. These optimization criteria estimate optimal adversarial manipulations such that adversarial cost function tends to relatively low value and classification error function yields a relatively high value.

Table 2 shows the results of statistical hypothesis testing across different attack performances, such as, CNN_o, CNN_m and CNN_s, under all the attack parameters values and datasets as mentioned in Figure 3 and Figure 4. Each row in Table 2 lists the attack parameter for statistical testing. Each column in Table 2 gives results of a two-sample t-test for each pair of CNN_o, CNN_m, CNN_s classifiers. The null hypothesis in each t-test states that the classifier performances are the same before and after adversarial manipulation. Assuming a normal distribution for the attack performances calculated over attack scenarios, the alternative hypothesis in each t-test states that across attack parameter settings (and adversarial datasets), the manipulated classifier has higher error than original classifier and the secure classifier has lower error than manipulated classifier.

Table 3 and Table 4 give the CNN defence performances calculated from f_1-score with varying (pos, neg) label pairs. Across the table rows, the attack parameters are fixed to target-dependent tuples, such as $s = 50$, $\lambda = 50$ and $N = 30$. Varying target class pos from 2 to 9 allows us to create different adversarial data manipulations on both the original data and the generated data. The defence performance of original CNN (CNN_o) and secure CNN (CNN_s) are our baseline for performance validation in Table 3 and Table 4. The defence performances of manipulated CNNs (CNN_m) in Table 3 and Table 4 are calculated on the original data distribution as well as the generated data distribution. CNN_o's defence performance is created on training data of either original data distribution or generated data distribution. By adversarially manipulating the corresponding testing data, CNN_m's

TABLE 2: Alternating Least Squares Attack Scenario: t-tests by varying parameters in Variational Stackelberg games. The small p-values from the statistical tests demonstrate the superiority of our model.

Attack Parameter	p-values from t-statistics for target class "5"			p-values from t-statistics for target class "Andy Lau"		
	CNN_o vs CNN_m	CNN_o vs CNN_s	CNN_m vs CNN_s	CNN_o vs CNN_m	CNN_o vs CNN_s	CNN_m vs CNN_s
Code size (s)	9.9×10^{-8}	3.6×10^{-8}	9.5×10^{-5}	1.9×10^{-4}	4.5×10^{-9}	4.1×10^{-8}
Cost weight (λ)	2.7×10^{-8}	1.8×10^{-7}	3.2×10^{-6}	6.5×10^{-9}	6.1×10^{-2}	1.4×10^{-7}
Annealing steps limit (N)	1.5×10^{-9}	1.1×10^{-8}	2.0×10^{-6}	3.4×10^{-8}	5.6×10^{-3}	1.4×10^{-6}

TABLE 3: MNIST Comparisions on the defence to adversarial Nash equilibrium attacks.

Classification error: Adversarial attack in Variational Stackelberg Game									Class Labels (pos, neg)
$CNN_{original}$	$CNN_{manipulated}$							CNN_{secure} $(Our\ method)$	
	CNN	$DCGAN$ [20]	$IWGAN$ [16]	$DeepFool$ [14]	$FGSM$ [2]	CNN_{GA} [30]	CNN_{SA} [30]		
1.12	10.01	40.16	39.41	32.06	34.66	25.93	25.81	6.44	(2,8)
1.54	54.74	47.12	39.84	46.15	37.71	27.88	28.76	5.11	(3,8)
3.86	24.16	26.18	26.75	26.59	26.57	28.58	26.82	6.33	(4,8)
1.87	8.78	50.68	53.31	56.71	35.91	27.94	28.49	5.31	(5,8)
1.21	12.41	39.41	36.16	47.19	35.58	25.77	26.16	5.39	(6,8)
4.89	23.91	28.78	27.48	27.03	27.28	27.65	27.81	4.82	(7,8)
2.53	16.97	41.21	37.22	54.88	36.85	26.86	27.34	3.58	(9,8)
t-statistics	1.9×10^{-2}	3.7×10^{-7}	6.8×10^{-7}	7.6×10^{-6}	2.0×10^{-9}	3.5×10^{-14}	4.6×10^{-14}	Base	

TABLE 4: VGGFace2 Comparisions on the defence to adversarial Nash equilibrium attacks.

Classification error: Adversarial attack in Variational Stackelberg Game									Class Labels (pos,neg)
$CNN_{original}$	$CNN_{manipulated}$							CNN_{secure} $(Our\ method)$	
	CNN	$DCGAN$ [20]	$IWGAN$ [16]	$DeepFool$ [14]	$FGSM$ [2]	CNN_{GA} [30]	CNN_{SA} [30]		
5.25	44.44	62.21	45.34	45.99	46.13	32.37	92.0	26.53	(Andy Lau, Jackie Chan)
20.19	37.71	73.33	56.31	26.45	33.81	22.07	61.34	13.09	(Zhang Jingchu, Jackie Chan)
20.11	35.19	27.64	31.13	32.64	34.91	33.88	26.27	16.66	(Ajay Devgan, Jackie Chan)
8.08	31.41	71.84	35.89	20.38	22.03	23.28	17.51	12.31	(Aishwarya Rai Bachchan, Jackie Chan)
21.92	32.62	81.94	31.75	26.27	32.17	33.01	84.51	11.74	(Amy Smart, Jackie Chan)
16.77	28.13	21.61	26.08	56.14	54.36	95.08	18.24	20.52	(Jada Pinkett Smith, Jackie Chan)
2.24	35.09	87.5	34.49	85.91	79.83	100	94.49	11.07	(Bruce Willis, Jackie Chan)
24.44	31.31	67.93	27.17	28.35	36.67	96.69	26.08	9.57	(Oprah Winfrey, Jackie Chan)
t-statistics	4.8×10^{-6}	1.1×10^{-4}	1.7×10^{-4}	7.3×10^{-3}	1.0×10^{-3}	8.1×10^{-3}	8.7×10^{-3}	Base	

defence performance is created. CNN_s's defence performance is created by training on adversarially manipulated training data and testing on adversarially manipulated testing data.

The original training data and original testing data is cross-validation data created from the MNIST database of handwritten digits [35] and VGGFace2 database of human faces [36]. The generated data is created from the outputs of Deep Convolutional Generative Adversarial Network (DCGAN) [20] and Improved Wasserstein Generative Adversarial Network (IWGAN) [16], which are generative adversarial networks (GANs) trained to produce images. The generated data is also created from existing adversarial examples in deep learning networks, i.e., DeepFool [14] and Fast Gradient Sign Method (FGSM) [2]; game theoretical adversaries genetic algorithm CNN_{GA} [30] and annealing algorithm CNN_{SA} [30]. The p-values from two-sample t-tests in the last row of Table 3 compare CNN_m's performances with CNN_s (which acts as the classification baseline for statistical comparisons).

The null hypothesis for t-test is that our adversarial manipulations are unable to attack existing adversarial classifiers where classifier's defence performance is calculated in terms of f_1-score of targeted class labels. The alternative hypothesis for t-test is that our adversarial manipulations are able to attack classifiers trained on existing sources for original data, generated data and adversarial data. Moreover, our secure classifier is robust to the proposed data manipulations in comparision to the existing adversarial classifiers.

Therefore, across multiple two-label two-player sequential variable-sum Stackelberg games played over both original data and generated data, CNN_m has greater error than CNN_o error in Table 3 and Table 4. At the same time CNN_s has lesser error than CNN_m. From low p-values ($<$0.05) in Table 3 and Table 4 we are able to reject the null hypothesis in t-tests comparing CNN_m with CNN_s. The errors thresholds $maxerror\_game$, $maxerror\_als$, $maxerror\_sa$, which manipulates training data in the game's iterations, allow us to control the amount of adversarial data that is injected into the testing data to achieve desired defence performance after game's convergence. On a given dataset, a desired value for $maxerror\_game$, $maxerror\_als$, $maxerror\_sa$ can be obtained by experimenting with parameter settings for λ, s, N, respectively. Varying the negative label neg is expected to create new values for the defence performances in Table 3, but not change the conclusions from t-tests over same set of positive labels pos.

From p-values of the t-tests, we also conclude that we are able to create attack data for the adversarial examples generated by GANs and adversarial networks. In comparision to the attack scenarios in Chivukula and Liu [30], our improved search and convergence criteria in the changed optimization problem are able to find better adversarial manipulations at the Nash equilibrium. These manipulations are able to attack the game theoretical adversarial learning proposed by Chivukula and Liu [30]. Finally we are then able to produce a more robust CNN_s classifier in comparision to existing literature.

6 Conclusion and Future Work

In this research, we aim to improve robustness of deep learning models via game theoretical modelling. We propose Stackelberg games to generate adversarial data that misleads CNNs classification result. We design adversarial payoff functions that optimize

the search for data manipulations on a latent data space represented by a Gaussian mixture model. We introduce payoff functions that are optimized by our search algorithms using simulated annealing and alternating least squares. After obtaining the optimal adversarial attack data from Nash equilibrium, we augument the training data with the adversarial data to produce CNNs secure to the proposed data manipulations. The CNN's misclassification performance is evaluated against existing attack models using statistical significance tests. Our experiments illustrate that our game-theoretical approach produces classifiers that are the most secure in defending various types of adversarial attacks.

In future work, we shall explore dependence between randomization in our adversarial manipulations and optimization in our game formulation. At present, the game theoretical stochastic optima (solving for adversarial data) are determined by the convergence of adversarial cost function rather than classification cost function. In future, we shall explore multilabel classification cost functions in a multiplayer strategy space of pure strategies as well as mixed strategies. Besides, we are also planning to apply our work to text and document data in natural language processing problems where the learning system may also be vulnerable to adversarial attacks.

REFERENCES

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of International Conference on Learning Representations*, 2014.

[2] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of International Conference on Learning Representations*, 2015.

[3] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. ACM, 2018.

[4] B. Li and Y. Vorobeychik, "Feature cross-substitution in adversarial classification," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014.

[5] N. Cesa-Bianchi and G. Lugosi, *Prediction and Playing Games*. Cambridge University Press, 2006.

[6] M. Ummels, "Stochastic multiplayer games: theory and algorithms," Ph.D. dissertation, RWTH Aachen University, 2011. [Online]. Available: http://darwin.bth.rwth-aachen.de/opus3/volltexte/2011/3451/pdf/3451.pdf

[7] Y. Zhou, M. Kantarcioglu, and B. Xi, "A survey of game theoretic approach for adversarial machine learning," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2019.

[8] T. Alpcan, B. I. P. Rubinstein, and C. Leckie, "Large-scale strategic games and adversarial machine learning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016.

[9] F. A. Oliehoek, R. Savani, J. Gallego-Posada, E. van der Pol, E. D. de Jong, and R. Gross, "Gangs: Generative adversarial network games," *CoRR*, vol. abs/1712.00679, 2017. [Online]. Available: http://arxiv.org/abs/1712.00679

[10] N. Papernot, P. McDaniel, A. Sinha, and M. P. Wellman, "Sok: Security and privacy in machine learning," in *2018 IEEE European Symposium on Security and Privacy (EuroS P)*, 2018.

[11] L. Yang, P. Li, Y. Zhang, X. Yang, Y. Xiang, and W. Zhou, "Effective repair strategy against advanced persistent threat: A differential game approach," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 7, pp. 1713–1728, July 2019.

[12] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017.

[13] S. Baluja and I. Fischer, "Learning to attack: Adversarial transformation networks," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *Proceedings of Conference on Computer Vision and Pattern Recognition CVPR*, 2016.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems (NIPS)*, 2014.

[16] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems 30*, 2017.

[17] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems 29*, 2016.

[18] C. Xiao, B. Li, J. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," in *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI*, 2018.

[19] J. Kos, I. Fischer, and D. Song, "Adversarial examples for generative models," in *Proceedings of 2018 IEEE Security and Privacy Workshops (SPW)*, 2018.

[20] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, 2015.

[21] J. Gilmer, R. P. Adams, I. J. Goodfellow, D. Andersen, and G. E. Dahl, "Motivating the rules of the game for adversarial example research," *CoRR*, 2018.

[22] N.-T. Tran, T.-A. Bui, and N.-M. Cheung, "Dist-gan: An improved gan using distance constraints," in *Proceedings of European Conference on Computer Vision (ECCV)*, 2018.

[23] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '04. New York, NY, USA: ACM, 2004, pp. 99–108.

[24] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, ser. KDD '05. New York, NY, USA: ACM, 2005, pp. 641–647. [Online]. Available: http://doi.acm.org/10.1145/1081870.1081950

[25] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proceedings of the 29th International Coference on International*

Conference on Machine Learning, ser. ICML'12. USA: Omnipress, 2012, pp. 1467–1474. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042573.3042761

[26] M. Brückner, C. Kanzow, and T. Scheffer, "Static prediction games for adversarial learning problems," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2617–2654, Sep. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2503308.2503326

[27] M. Kantarcioglu, B. Xi, and C. Clifton, "Classifier evaluation and attribute selection against active adversaries," *Data Min. Knowl. Discov.*, vol. 22, no. 1-2, pp. 291–335, Jan. 2011. [Online]. Available: http://dx.doi.org/10.1007/s10618-010-0197-3

[28] W. Liu and S. Chawla, "Mining adversarial patterns via regularized loss minimization," *Mach. Learn.*, vol. 81, no. 1, pp. 69–83, Oct. 2010.

[29] F. Wang, W. Liu, and S. Chawla, "On sparse feature attacks in adversarial learning," in *2014 IEEE International Conference on Data Mining*, Dec 2014, pp. 1013–1018.

[30] A. Chivukula and W. Liu, "Adversarial deep learning models with multiple adversaries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 6, pp. 1066–1079, June 2019.

[31] Z. Yin, F. Wang, W. Liu, and S. Chawla, "Sparse feature attacks in adversarial learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 06, pp. 1164–1177, jun 2018.

[32] Y. Zhou and M. Kantarcioglu, "Modeling adversarial learning as nested stackelberg games," in *Advances in Knowledge Discovery and Data Mining*, J. Bailey, L. Khan, T. Washio, G. Dobbie, J. Z. Huang, and R. Wang, Eds. Cham: Springer International Publishing, 2016, pp. 350–362.

[33] M. Simaan and J. B. Cruz, Jr., "On the stackelberg strategy in nonzero-sum games," *J. Optim. Theory Appl.*, vol. 11, no. 5, pp. 533–555, May 1973.

[34] A. Bressan, "Noncooperative differential games," *Milan Journal of Mathematics*, vol. 79, no. 2, pp. 357–427, 2011.

[35] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database," *URL http://yann. lecun. com/exdb/mnist*, 1998.

[36] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "Vggface2: A dataset for recognising faces across pose and age," in *International Conference on Automatic Face and Gesture Recognition*, 2018.

[37] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*. ACM, 2012, pp. 1097–1105.

[38] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016.

[40] X. Hou, L. Shen, K. Sun, and G. Qiu, "Deep feature consistent variational autoencoder," in *2017 IEEE Winter Conference on Applications of Computer Vision, WACV 2017, Santa Rosa, CA, USA, March 24-31, 2017*, 2017.

Aneesh Sreevallabh Chivukula received MS(by Research) in Computer Science and Engineering from International Institute of Information Technology(IIIT), Hyderabad, India. He is currently a PhD student with the Advanced Analytics Institute, School of Computer Science, the University of Technology Sydney, Australia. He has been working on computational data science at startup companies and research labs. His research interests are in computational algorithms, intelligent systems, data mining and big data analytics.

Xinghao Yang received the B.Eng. degree in electronic information engineering and M.Eng. degree in information and communication engineering from the China University of Petroleum (East China), Qingdao, China, in 2015 and 2018, respectively. Currently, he is a PhD student in Advanced Analytics Institute, University of Technology Sydney, Australia. His research interests include multi-view learning and adversarial machine learning with publications on information fusion and information sciences.

Wei Liu (M'12) received the PhD degree in computer science from the University of Sydney, Australia. He is the Data Science Program Leader and a Senior Lecturer in the Advanced Analytics Institute, School of Computer Science, the University of Technology Sydney, Australia. He works in the areas of machine learning and data mining and has published more than 80 papers in research topics of tensor factorization, adversarial learning, graph mining, causal inference, recommendation systems, and anomaly detection. He has won best paper awards.

Tianqing Zhu (M'11) received the B.Eng. and M.Eng. degrees from Wuhan University, China, in 2000 and 2004, respectively, and the Ph.D. degree in computer science from Deakin University, Australia, in 2014. She served as a Lecturer with Wuhan Polytechnic University, China, from 2004 to 2011. She is currently a senior lecturer at the Centre for Cyber Security and Privacy, School of Computer Science, University of Technology Sydney, Australia. Her research interests include privacy preserving, data mining, and network security. Dr. Tianqing has received the Best Student Paper Award in PAKDD 2014.

Wanlei Zhou (SM'09) received the B.Eng. and M.Eng. degrees from the Harbin Institute of Technology, Harbin, China, in 1982 and 1984, respectively, the Ph.D. degree from Australian National University, Canberra, ACT, Australia, in 1991, all in computer science and engineering, and the D.Sc. degree from Deakin University, Melbourne, VIC, Australia, in 2002. He is currently a Professor and the Head of School of Computer Science at the University of Technology Sydney. He served as a Lecturer with the University of Electronic Science and Technology of China, a System Programmer with Hewlett Packard, Boston, MA, USA, and a Lecturer with Monash University, Melbourne, VIC, Australia, and the National University of Singapore, Singapore. He has published over 300 papers in refereed international journals and refereed international conferences proceedings. His research interests include distributed systems, network security, bioinformatics, and e-Learning. Dr. Wanlei was the General Chair/Program Committee Chair/Co-Chair of a number of international conferences, including ICA3PP, ICWL, PRDC, NSS, ICPAD, ICEUC, and HPCC.