Adversarial Deep Learning Models with Multiple Adversaries

Aneesh Sreevallabh Chivukula and Wei Liu, *Member, IEEE*

Abstract—We develop an adversarial learning algorithm for supervised classification in general and Convolutional Neural Networks (CNN) in particular. The algorithm's objective is to produce small changes to the data distribution defined over positive and negative class labels so that the resulting data distribution is misclassified by the CNN. The theoretical goal is to determine a manipulating change on the input data that finds learner decision boundaries where many positive labels become negative labels. Then we propose a CNN which is secure against such unforeseen changes in data. The algorithm generates adversarial manipulations by formulating a multiplayer stochastic game targeting the classification performance of the CNN. The multiplayer stochastic game is expressed in terms of multiple two-player sequential games. Each game consists of interactions between two players—an intelligent adversary and the learner CNN—such that a player's payoff function increases with interactions. Following the convergence of a sequential noncooperative Stackelberg game, each two-player game is solved for the Nash equilibrium. The Nash equilibrium finds a pair of strategies (learner weights and evolutionary operations) from which there is no incentive for either learner or adversary to deviate. We then retrain the learner over all the adversarial manipulations generated by multiple players to propose a secure CNN which is robust to subsequent adversarial data manipulations. The adversarial data and corresponding CNN performance is evaluated on MNIST handwritten digits data. The results suggest that game theory and evolutionary algorithms are very effective in securing deep learning models against performance vulnerabilities simulated as attack scenarios from multiple adversaries.

Index Terms—Supervised learning, data mining and knowledge discovery, evolutionary learning, adversarial learning, deep learning, stochastic optimization, game theory

1 Introduction

To learn mathematical patterns, machine learning methods make assumptions on the data distributions for training and testing the learning algorithm. In this paper we design an algorithm that generates a testing data distribution which is non-stationary with respect to the training data distribution. Designing robust data mining models, computing systems and machine learning algorithms for non-stationary data analytics is the goal of adversarial learning. Adversarial learning has application in areas such as spam filtering, virus detection, intrusion detection, fraud detection, biometric authentication, network protocol verification, computational advertising, recommender systems, social media web mining and performance modelling of complex systems [1], [2].

Adversarial learning algorithms are specifically designed to exploit vulnerabilities in a given machine learning algorithm. These vulnerabilities are simulated by training the learning algorithm under various attack scenarios and policies. The attack scenarios are assumed to be formulated by an intelligent adversary [3]. The optimal attack policy is formulated to solve one or many optimization problems over one or many attack scenarios. A learning algorithm designed over adversarial settings becomes robust to such vulnerabilities in the training and testing data distributions. The various adversarial learning algorithms differ in assumptions regarding the adversary's knowledge, security violation, attack strategies and attack influence [4].

We formulate and customize objective functions and search algorithms for adversarial learning where the learner is assumed to be a deep neural network. We then derive adversarial manipulations and defence mechanisms for deep learning algorithms. Deep learning refers to a class of neural network algorithms with many stages of nonlinear information processing in hierarchical architectures exploited for pattern classification and feature learning [5]. Deep learning research aims at discovering machine learning algorithms at multiple levels of data abstractions. In high dimensional data, deep learning has been found to be susceptible to adversarial examples [6]. Such adversarial examples can be crafted by prior knowledge, observation, and experimentation on the network layers and loss functions in the deep learning model.

The intuition for our objective function is derived from game theory actions and moves. We assume that the attack scenarios in adversarial learning can be modelled as the moves made by a learning algorithm and countermoves made by an intelligent adversary. Game Theory is the study of interactions or games between independent self-interested agents or players. Each player has a set of associated strategies/moves/actions that optimize a payoff function or

• *The authors are with the Advanced Analytics Institute, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia. E-mail: AneeshSrivallabh.Chivukula@student.uts.edu.au, Wei.Liu@uts.edu.au.*

Manuscript received 14 July 2017; revised 2 Mar. 2018; accepted 20 June 2018. Date of publication 28 June 2018; date of current version 26 Apr. 2019.
(Corresponding author: Wei Liu.)
Recommended for acceptance by M. Wang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2018.2851247

TABLE 1
Adversarial Algorithms Comparision by Design

Adversarial algorithm	Adversary's knowledge	Attack strategy	Search algorithm	Convergence Conditions
Classifier ensembles [7]	Training features	Reorder features by importance for discriminant function	Randomized sampling	Ensemble size, feature subset size
Feature weighting [8]	Training features	Addition/deletion of binary features	Feature bagging	Number of base models
SVM : inputs [9]	Gradient of loss	Train noise injection	Gradient ascent	Change in test error
SVM : labels [10]	Training labels	Label noise injection	Gradient ascent	Support vectors from linear and quadratic programming
Adversarial networks : GAN [6]	Training and testing data	Linear perturbation on training data	Backpropagation with L-BFGS	Early stopping on adversarial validation set error
Adversarial networks : DNN [11]	Testing data	Observe DNN outputs given inputs chosen by the adversary	Jacobian-based dataset augmentation	Early stopping on adversarial validation set error
Adversarial networks : DAE [12]	Testing data	Gaussian additive noise	Stacking DAEs into a feed forward neural network	Training error
Game theory : support vector machines [13]	Training features	Delete different features from different data points	Quadratic programming	Training error subject to regularization terms
Game theory : deep learning (Our method)	Training data	Move positive samples towards negative samples	Evolutionary algorithm	Nash equilibrium

utility function. The key idea in game theory is that of an equilibrium state from which none of the players have any incentive to deviate. A two-player game where a single follower acts in response to the moves of a single leader is called a Stackelberg game. It is said to converge onto the equilibrium state called Nash equilibrium.

Our adversarial algorithm proposes a game between two players—a data miner or learner and an intelligent adversary or adversary. The interactions between the learner and adversary are modelled as a two-player sequential Stackelberg zero-sum game. In our game, the adversary is the leader and the learner is the follower. The learner retrains the model after the adversary's attack. The payoff function for each player is specified in terms of objective functions simulating the adversary's attack process and learner's learning processes. The attack processes specify the adversary's constraints and optimal attack policy. The learning processes specify the learner's gain and adversary's gain under the optimal policy. The optimal attack policy is formulated in terms of stochastic optimization operators and evolutionary computing algorithms.

Following are the major contributions of this research.

- We formulate the problem of finding attack scenarios in adversarial learning and defence mechanisms in deep learning as a maxmin optimization problem rooted in game-theoretic learning models.
- We formulate two-player games and multiplayer games simulating adversarial manipulations in terms of adversarial cost and learning error on the training and testing data distributions.
- We represent the solution of adversarial manipulations by stochastic operators and game strategies in evolutionary algorithms. We evaluate adversarial manipulations with a tensor-based fitness function and corresponding payoff functions in the game.
- Our algorithms can adapt to continuous adversarial data manipulations unlike most of the existing adversarial learning algorithms. We do not assume the adversary knows anything about the deep network structure which is close to real-life settings.
- On the MNIST data, we demonstrate the effectiveness of the proposed adversarial manipulations in both the two-player and multiplayer games. Furthermore, both convolutional neural networks and generative adversarial networks are shown to be vulnerable to the adversarial manipulations.

The paper starts with related work in Section 2 comparing the new approach with existing approaches. The stochastic game and corresponding sequential game is formulated in Section 3. The pseudocode for the proposed sequential game and the experiments for the proposed stochastic game are presented in Sections 3 and 6 respectively. The paper ends in Section 7 with a summary of current work and future work.

2 Related Work

The existing adversarial learning algorithms are summarized in Table 1 by algorithm design. The table's columns list the various features to compare the adversarial learning algorithms. The table's rows list the various algorithms under comparison. The algorithms are compared in terms of adversary's knowledge, attack strategy, search algorithm and convergence conditions. The "adversary's knowledge" is the semantic information of the adversary. The "security violation" is the purpose of the adversarial attack setting. The "attack strategy" is the attack scenario under which the adversary operates. The "search algorithm" is the algorithm used to find an optimal solution. The "convergence conditions" are the search criteria for creating adversarial data. Our algorithm is termed "Game theory: deep learning".

2.1 Adversarial Security Mechanisms

As shown in Table 1, the existing adversarial learning algorithms and applications can be classified by the learner's defence mechanisms and adversary's attack scenarios [4], [7], [14], [15]. The learner's defence mechanisms have been proposed by designing secure learning algorithms [4], multiple classifier systems [7], privacy-preserving machine learning [14] and the use of randomization or disinformation to mislead the adversary [15].

Biggio et al. [4] discuss the learner's defence mechanism in terms of an empirical framework extending the model selection and performance evaluation steps of pattern classification [16]. The framework recommends training the learner for "security by design" rather than "security by obscurity". The additional steps validate the defence mechanisms proposed in the case of both generative learning and discriminative learning under attack. Depending on the goal, knowledge and capability of the adversary, these steps are classified in terms of attack influence, security violation and attack specificity.

Our algorithm has causative attack influence, integrity security violation and targeted attack specificity. It causes a slight change to the data distributions simulated by stochastic optimization and randomized sampling methods. The optimization problem converges into a solution computed at the Nash equilibrium in the game. From the adversary's standpoint, the equilibrium solution is a local optimum in worst-case attack scenarios and a global optimum in best-case attack scenarios. The strength and relevancy of the attack scenario is determined by the performance of the deep learning model under attack.

2.2 Generative Adversarial Networks

Goodfellow et al. [6] argue for the need to have an adversarial training procedure with the objective to minimize the worst case error when the data is perturbed by an adversary. Goodfellow et al. [6] propose a minmax game between two deep learning networks called Generative Adversarial Networks (GANs). A variety of generative methods are available to create the perturbation between training and testing data distributions [17]. Radford et al. [18] propose a stable GAN called DCGAN. Gulrajani et al. [19] design IWGAN which undertakes a theoretical analysis of the generative learning process. Berthelot et al. [20] propose BEGAN with a new loss function in the training algorithm. Chen et al. [21] propose InfoGAN which uses generative learning models for unsupervised representation learning.

Insofar as the learner's defence mechanisms are concerned, our game formulation is similar to the GAN game formulation. However, the objective of our research is to simulate a real adversarial attack scenario on two-label classification model in terms of cost to the adversary. We seek to increase the classification performance when the data distribution is changed with a malicious intent. By contrast, the objective of GAN is to generate synthetic data that is indistinguishable from the original data. Our objective function has cost and error terms defining the attack scenarios in adversarial data settings. By contrast, the objective function in GAN is defined in terms of the loss functions of deep neural networks underlying the given training and testing data distributions.

In a minmax game formulation, we seek to create the datasets for attack scenarios in a discriminative learning model and supervised learning problem while GAN addresses a generative learning model and unsupervised learning problem. Furthermore, the generator is the leader of the game in minmax formulation for GAN, whereas in our minmax formulation an intelligent adversary leads the game. While searching for the Nash equilibrium in a minmax game, GANs solve a convex optimization problem with gradient-based optimization algorithms whereas we solve a non-convex stochastic optimization problem with evolutionary learning algorithms. Thus, we are able to estimate the best cost for the adversary in effecting the adversarial attack.

2.3 Game-Theoretic Learning Models

Stochastic games defined on a strategy space have been used to generate adversarial examples [22]. The strategy space is defined in terms of two or more adversaries' actions and corresponding payoff functions. Each adversary can engage one or more learners in a game and vice versa. From the learner's standpoint, adjusting parameters is computationally less expensive than building a new model that is robust to adversarial manipulation. From the adversary's standpoint, the attack scenarios can be characterized by the stochastic optimization parameters estimated in the game. A game ends in an equilibrium with payoffs to each player based on their objectives and actions. The learner has no incentive to play a game that leads to too many false positives with too little increase in true positives. The adversary has no incentive to play a game that increases the utility of false negatives not detected by the learning algorithm. At equilibrium, the adversary is able to find testing data that is significantly different from the training data whereas the learner is able to update its model for new threats from adversarial data.

Globerson and Roweis [13] discuss a classification algorithm with a game theoretic formulation. The proposed algorithm is robust to feature deletion according to a minmax objective function optimized by quadratic programming. In Liu and Chawla [23], the interactions between an adversary and data miner are modelled as a two-player sequential Stackelberg zero-sum game where the payoff for each player is designed as a regularized loss function. The adversary iteratively attacks the data miner using the best possible strategy for transforming the original training data. The data miner independently reacts by rebuilding the classifier based on the data miner's observations of the adversary's modifications to the training data. The game is repeated until the adversary's payoff does not increase or the maximum number of iterations is reached. Liu et al. [24] propose an extension to Liu and Chawla [23] where a one-step game is used to reduce the computing time of the minmax algorithm. The one-step method converges to Nash equilibrium by utilizing Singular Value Decomposition (SVD). Yin et al. [25] formulate a bi-level optimization problem from a non-zero sum game on adversarial data transformations. The game experiments with sparse regularizers for designing robust classification objectives.

To derive the payoff functions in the game, we assume that the adversary has no knowledge of either the deep neural network layers or loss functions in the deep learning model. The proposed minmax problem is solved without making assumptions on the training/testing data distributions. The strategy space for algorithmic randomization and data manipulation in the game is determined by the stochastic operators in evolutionary algorithms defining the attack scenarios.

2.4 Stochastic Optimization

Evolutionary Algorithms (EA) have been used in stochastic optimization to generate rule-based data mining models with attribute interactions [26]. The EA-based stochastic search and optimization algorithms are Evolutionary Programming (EP), Evolutionary Strategies (ES), Genetic Algorithms (GA), Differential Evolution (DE), Estimation of Distribution Algorithm (EDA) and Swarm Intelligence (SI) algorithms [27], [28].

In our adversarial algorithm, the search and optimization algorithm is either a genetic algorithm or a simulated annealing algorithm. The adversarial data samples are generated by the selection, crossover, mutation search operators in the genetic algorithm and the annealing search operator in the simulated annealing algorithm. By using probabilistic hill climbing algorithms over Markov chains in multivariate models, the current search operators can be extended to define explicit probabilistic distributions performing a complex neighbourhood search for the candidate solutions [29].

3 Game Formulation

In this section, we discuss the problem formulation for the proposed adversarial learning algorithm. The stochastic multiplayer game is formulated in terms of multiple two-player sequential games.

3.1 Sequential Game Formulation

The training algorithm simulates the adversarial learning as a constant sum Stackelberg game between two players. The two players are called Leader (L) and Follower (F). The leader initiates the game by making the first action/move/play. In our algorithm, the adversary is the leader and the learner is the follower. In a constant sum game, the learner's loss is assumed to be the adversary's gain and vice versa.

Each player is associated with strategy spaces A and W for L and F respectively. The strategy space is a choice of moves available to each player. The outcome of a strategy is determined by the player's payoff function J_L and J_F. For a given observation of $w \in W$, the best strategy $\alpha^* \in A$ for the leader is

$$\alpha^* = argmax_{\alpha \in A} J_L(\alpha, w). \tag{1}$$

Similarly for L's move α, F's best strategy is

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

Moreover, the sum of payoff functions J_L for the adversary and J_F for the classifier is assumed to sum to a constant profit Φ. This allows us to rewrite the expression for w^* in terms of J_L

$$w^* = argmax_{w \in W} \Phi - J_L(\alpha, w) = argmin_{w \in W} J_L(\alpha, w). \tag{3}$$

Combining Equation (1) with Equation (3) we formulate the following maxmin problem in the game.

$$Maxmin : (\alpha^*, w^*) = argmax_{\alpha \in A} J_L(\alpha, argmin_{w \in W} J_L(\alpha, w)). \tag{4}$$

We train a Convolutional Neural Network (CNN) as the learner. With knowledge of only the learner's classification error, the adversary is assumed to target the true positives. In each iteration of the game, the learner trains the weights w in the CNN layers for the input α presented by the adversary. The adversary then adapts the data manipulations to the weights trained by the CNN. Thus, each player's move is based on the opponent's last play. The game is initiated by the adversary. Thus the adversary is the leader L and the learner is the follower F. Each game iteration is composed of the moves made by L and F.

Using an evolutionary algorithm, the adversary searches for data manipulations that maximize classification error $error(w)$. The fitness function in evolutionary algorithm $J_L(\alpha, w)$ is defined to be the adversary's payoff function. The fitness function increases with iterations of the game. The game converges when the adversary does not see an increase in the payoff function or the maximum number of iterations is reached. The game convergence criteria depend on the search and optimization criteria of the evolutionary algorithm used in each game iteration. The game converges to an adversarial data manipulation α^* on the learner with weights w.

For labelled input training data X_{train}, X_{test} available during the game, the adversary searches for a move α that maximizes the following payoff function or fitness function $J_L(\alpha)$ where $error$ is the classification error as measured by $recall$ for the current adversarial data. The term $cost$ is the ℓ_2 norm for the current α.

$$J_L(\alpha, w) = 1 + \lambda * error(w) - cost(\alpha) \tag{5}$$

$$error(w) = 1 - recall(w) \tag{6}$$

$$cost(\alpha) = ||\alpha||_2. \tag{7}$$

The negative $cost(\alpha)$ term in Equation (5) ensures that the adversary minimizes changes to the current α while maximizing the positive $error(w)$ term. By definition of the fitness function, $error(w)$ is maximized by minimizing the corresponding $recall(w)$. $recall(w)$ is computed for each iteration of the game on the manipulated training data $X_{train} + \alpha$ so that the α which gives the maximum value for $J_L(\alpha, w)$ is selected for subsequent iterations in the game. $cost(\alpha)$ is enhanced by an weighting term λ that is empirically evaluated for each dataset. A constant 1 is then added to $J_L(\alpha)$ to ensure a positive fitness function in the evolutionary algorithm.

Therefore, from a theoretical standpoint, we characterize the statistical difference between the input training data distribution X_{train} and adversarial testing data distribution $X_{test} + \alpha^*$ in terms of the adversary's cost $cost(\alpha)$ and the learner's error $error(w)$. During the game's iterations, the manipulation of the training data distribution X_{train} into $X_{train} + \alpha$ allows us to find the α that maximizes the adversary's payoff $J_L(\alpha, w)$. After game convergence, the manipulation of the testing data distribution X_{test} into $X_{test} + \alpha^*$ allows us to find the α^* that minimizes learner's payoff $J_F(\alpha, w)$. Equation (5) is solved for α^*—the additive pixel level manipulations generating the adversarial manipulations. α^* allows us to find an adversarial testing data distribution $X_{test} + \alpha^*$ that is non-stationary with respect to the original training data distribution X_{train}.

Various multiplayer multi-label game formulations would allow us to optimize various adversarial manipulations on the input data distribution. Equation (5) can also be solved for more types of adversarial manipulations whose cost of generation $cost(\alpha)$ is characterized by structured adversarial manipulations on complex data structures like tensors and graphs increasingly found in big data models.

From an application standpoint, we observe that the adversary's interest is in converting illegitimate data to legitimate data with a minimum of changes and not vice versa. To account for this objective, we assume the learner's error to be classification recall. By converting true positives to either false positives or false negatives, the adversary tries to reduce the true positives in the learner's performance regardless of the changes to the true negatives. The adversary's attack scenarios are expressed in terms of the parameter settings in the evolutionary algorithms. Each different parameter setting allows us to converge onto a different adversarial data manipulation on the testing data distribution.

From the perspective of improving the robustness of deep learning networks, our formulation generates the manipulated data distributions maximizing the error of the deep network and measuring the cost of the corresponding attack scenarios generating data manipulations over the training data in adversarial learning. A successful attack scenario allows us to estimate a manipulated data distribution that leads to statistically significant testing error. The estimation of neural networks weights on such data distributions is the area of our research and investigation.

3.2 Stochastic Game Formulation

In this section, we generalize Equation (5) for multiple adversaries participating as players in the game. At the end of the game, we propose a learner that is robust to the adversarial data generated by each adversary.

Suppose a set L of M adversaries $L = \{L_1, L_2, L_3, \dots, L_M\}$ associated with the strategy space A are attacking the single learner F associated with the strategy space W. The outcome of all M adversaries strategy set $\mathbb{A}_S = \{\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_M\}$ is determined by the vector $\boldsymbol{J_L}$ of M adversary payoff functions $\boldsymbol{J_L} = \{J_{L_1}, J_{L_2}, J_{L_3}, \dots, J_{L_M}\}$ such that J_{L_i} corresponds to α_i where $i = 1, 2, 3, \dots, M$. For a given best observation of $w^* \in W$ and best estimate of $\alpha^* \in A$, under the same assumptions, we can generalize the maxmin problem in Equation (4) to the following Equation (8) for a multiplayer constant sum stochastic game.

$$Maxmin : (\mathbb{A}_S^*, w^*) = argmax_{\mathbb{A}_S \in A} \boldsymbol{J_L}(\mathbb{A}_S, argmin_{w \in W} \boldsymbol{J_L}(\mathbb{A}_S, w)). \tag{8}$$

In this research, we assume that the vector $\mathbb{A}_S$ is the set of its component scalar α_i where $i = 1, 2, 3, \dots, M$, so that the multiplayer stochastic game can be simplified as many two-player sequential games. Each two-player game is played as a sequential game following Equations (4) and (5). The output of all the games is a set $\mathbb{A}_S^*$.

Upon convergence, each game i outputs adversarial data manipulation α_i^* where $i = \{1, 2, 3, \dots, M\}$. Each α_i^* is added to the original training data distribution X_{train} and original testing data distribution X_{test} to create final adversarial manipulations $X_{train} + \mathbb{A}_S^*$ and $X_{test} + \mathbb{A}_S^*$.

Definition 1. $X_{train} + \mathbb{A}_S^* = \{\cup_{i=1}^M \{X_{train} + \alpha_i^*\}\}$

Definition 2. $X_{test} + \mathbb{A}_S^* = \{\cup_{i=1}^M \{X_{test} + \alpha_i^*\}\}$

We choose a Convolutional Neural Network as the learner. The CNN architecture's input layers and output layer are described in Krizhevsky et al. [30] and available in the Tensorflow[1] as the CIFAR10 model. The CNN has input layers consisting of convolution layers, maxpooling layers, regularization layers and activation units. The CNN has output layer of the softmax probability distribution function. The overall loss function of the learner is defined by the CNN's input and output layers.

From multiplayer game output $\mathbb{A}_S^*$, we define the following CNN models corresponding to manipulated training data distribution $X_{train} + \mathbb{A}_S^*$ and testing data distribution $X_{test} + \mathbb{A}_S^*$. X_{train} is sampled from a given MNIST database $X_{original}$ as well as a Generative Adversarial Network (GAN) output $X_{generated-gan}$.

Definition 3.

$$\begin{aligned} &CNN_{original} = CNN(X_{train}, X_{test}),\\ &X_{train} = Sample(X_{original}),\\ &X_{test} = Sample(X_{original}) \end{aligned}$$

Definition 4.

$$\begin{aligned} &CNN_{manipulated-cnn} = CNN(X_{train}, X_{test} + \mathbb{A}_S^*),\\ &X_{train} = Sample(X_{original}),\\ &X_{test} = Sample(X_{original} + \mathbb{A}_S^*) \end{aligned}$$

Definition 5.

$$\begin{aligned} &CNN_{manipulated-gan} = CNN(X_{train}, X_{test} + \mathbb{A}_S^*),\\ &X_{train} = Sample(X_{generated-gan}),\\ &X_{test} = Sample(X_{original} + \mathbb{A}_S^*) \end{aligned}$$

Definition 6.

$$\begin{aligned} &CNN_{secure} = CNN(X_{train} + \mathbb{A}_S^*, X_{test} + \mathbb{A}_S^*),\\ &X_{train} = Sample(X_{original} + \mathbb{A}_S^*),\\ &X_{test} = Sample(X_{original} + \mathbb{A}_S^*) \end{aligned}$$

3.3 Stochastic Game Illustration

Fig. 1 illustrates the learning process in the game formulation as a flow chart. The $CNN_{original}$ is trained on training data X_{train} and evaluated on testing data X_{test} given as 'learner performance' in the experiments described in Section 6. Fig. 1 illustrates a two-player game. The game has moves executed by each of the adversaries and the learner during each interaction. In these moves, an adversary targets the learner by the adversarial sample produced from the evolutionary operators. The learner then adapts the deep learning operators for the adversarial data by retraining the CNN on the new cross-validation sample.

A set L of M adversaries $L = \{L_1, L_2, L_3, \dots, L_M\}$ targets this performance by engaging the CNN in multiple

1. https://www.tensorflow.org/tutorials/deep_cnn#cifar-10_model

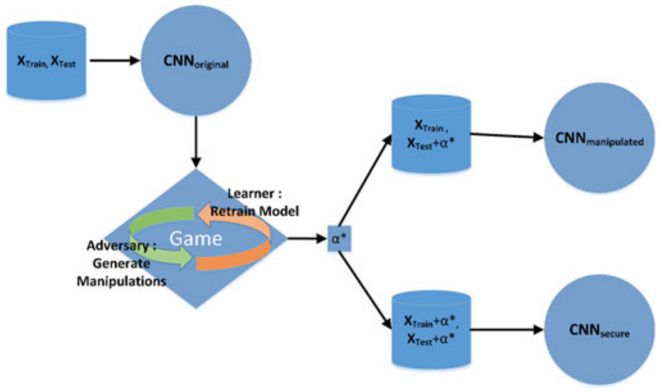


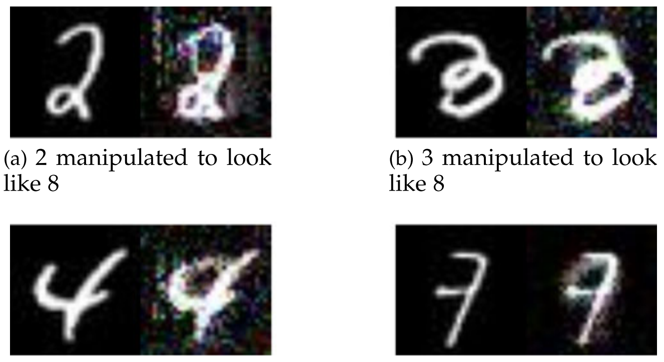
Fig. 1. A flow chart illustrating the benefits of a game theoretic learner. The two-player game is played by a single adversary and one Learner. The game produces a final deep learning network CNN_{secure} that is better equipped to deal with the adversarial manipulations than the initial deep learning network $CNN_{original}$.

two-player sequential games. In each two-player game, the CNNs trained on the original and generated data samples and tested on the adversarial data are $CNN_{manipulated-cnn}$, $CNN_{manipulated-gan}$ respectively. All these CNNs are given under the umbrella term 'manipulated learner performance' in the experiments in Section 6. We find that $CNN_{manipulated-cnn}$ as well as $CNN_{manipulated-gan}$ are significantly worse performing than the original CNN $CNN_{original}$ trained on the original training and testing data (X_{train}, X_{test}). Thus we conclude adversarial manipulation succeeds in attacking the learner. A new Convolutional Neural Network CNN_{secure} is then retrained on $(X_{train} + \mathbb{A}^*_S, X_{test} + \mathbb{A}^*_S)$ to adapt to adversarial manipulations. It is given as 'secure learner performance' in the experiments described in Section 6. CNN_{secure} is our proposed model. It is found to be better than the manipulated CNN's $CNN_{manipulated-cnn}$ and $CNN_{manipulated-gan}$.

Therefore, we conclude that the new CNN_{secure} has successfully adapted to adversarial data generated by multiple adversaries while the given $CNN_{original}$ is vulnerable to each adversarial manipulation α^*_i generated by each adversary L_i playing a game i on the given training/testing data distributions. Our algorithm is able to find a data sample that affects the performance of a CNN. The CNN that is able to recover from our adversarial attack is better equipped to deal with unforeseen changes in the underlying data distribution. The game between adversary and learner allows us to produce adversarial data manipulations for a CNN trained on the underlying data distribution.

3.4 Illustrative Examples

Fig. 2 shows examples of data transformations on positive labels that appear to be negative labels in the adversarial algorithm. Some of the transformations that avoid detection are adding and deleting pixels, and changing the shape and size of the image. In Fig. 2a, the handwritten digit 2 has been manipulated to look like a 8 by adding pixels. In Fig. 2b, the handwritten digit 3 has been manipulated to look like 8 by changing the thickness and shape. In Fig. 2c, the handwritten digit 4 has been manipulated to look like 9 by adding pixels and deleting pixels. In Fig. 2d, the handwritten digit 7 has been manipulated to look like 9 by adding pixels but not changing the shape.



(a) 2 manipulated to look like 8

(b) 3 manipulated to look like 8

(c) 4 manipulated to look like 9

(d) 7 manipulated to look like 9

Fig. 2. Examples of transformed images found at Nash equilibrium in a Stackelberg game. To avoid detection, the adversary adds pixels in (a) and (d), and changes shape in (b) and (c).

4 Stochastic Game Algorithm

In this section we discuss the adversarial learning algorithm with a set L of M adversaries participating in a stochastic game. Each adversary L_i participates in a sequential two-player game i with the CNN to output α^*_i according to either Algorithms 2 or 7.

Algorithm 1 solves for $\mathbb{A}^*_S$ in Equation (8). As input, the algorithm requires the labelled training data X_{train} and labelled testing data X_{test} Each record in the training and testing data is a tensor with pixel values representing an image. A variable M determines the number of adversaries participating in the multiplayer game. M is set to 1 for the two-player game. The variable *gametype* allows each adversary in a multiplayer game to choose between GA and SA as the optimization procedure for a two-player game.

The cumulative effect of all the adversaries attack is validated by datasets $X_{train} + \mathbb{A}^*_S$ and $X_{test} + \mathbb{A}^*_S$ following Definitions 1 and 2 respectively. In Section 6.5 and Table 6, these datasets are used to validate the CNN F1-score performance according to Definitions 3, 4, 5 and 6.

5 Sequential Game Algorithm

The pseudocode for a two-player sequential game i with output α^* is discussed in Sections 5.1 and 5.2. By assuming that each sequential game i is independent of the remaining games in the stochastic game with M adversaries, we drop game number i in the sequential game pseudocode. The experiments in Sections 6.3, 6.2 and Table 5 validate the sequential game. Section 6.2. Section 6.4 provides a conclusion to these experiments.

5.1 Genetic Algorithm

Algorithm 2 gives the training algorithm for the two-player sequential game that takes into consideration adversarial attacks. As input, Algorithm 2 requires the training data X_{train}. Each example in the input data is a three dimensional tensor of RGB pixel values. Algorithm 3 calculates the fitness function values for candidate solutions. Algorithms 4, 5, 6 give the genetic operators used by Algorithm 2 to search for candidate solutions.

Algorithm 2 initializes the game by training the CNN on Line 2 to store the weights to disk. The fitness function values are computed on Line 5 for each randomly initialized α. α belongs to a genetic population $\alpha_{population}$ operating on the input data X_{train}. $\alpha_{population}$ is randomly initialized around the mean of the positive class. It is assigned to $population$ on Line 4. A variable $maxpayoff$ is used to keep track of the adversary's current payoff in the current iteration of the game from Line 7 to Line 25. The if condition on Line 11 ensures that the algorithm converges when $maxpayoff$ does not exceed the current payoff $currpayoff$ by a small number 0.0001. In each game iteration, the current α_{curr} which gives the best fitness value is selected on Line 8. On Line 9, the CNN is retrained to react to attack $X_{train} + \alpha_{curr}$. The variable $maxpayoff$ is updated on Line 13 if α_{curr} satisfies the game convergence criteria.

Algorithm 1. Multiplayer Game with Multiple Adversaries

```
Input:
1: Labelled training data $X_{train}$, Labelled testing data $X_{test}$, Number of adversaries $M$, Optimization flag variable $gametype$
Output:
2: Adversarial manipulations $\mathbb{A}_S^*$, Attack performance F1-score$_{manipulated}$, Retraining performance F1-score$_{secure}$
3: for $i \in [1..M]$ do ▷ Iterate $i$ on M adversaries choosing either GA or SA as flag variable $gametype$
4:   Begin
5:     if $gametype$ == GA then
6:       $\alpha_i^*$ = twoplayergame-ga($X_{train}$)
7:     if $gametype$ == SA then
8:       $\alpha_i^*$ = twoplayergame-sa($X_{train}$)
9:   End
10: $\mathbb{A}_S^* = \{\alpha_i^*\}, i = \{1, 2, 3, \dots, M\}$
11: Generate $X_{train} + \mathbb{A}_S^*$ from Definition 1
12: Generate $X_{test} + \mathbb{A}_S^*$ from Definition 2
13: Train CNN on $X_{train}$
14: Calculate F1-score$_{manipulated}$ by testing CNN on $X_{test} + \mathbb{A}_S^*$
15: Train CNN on $X_{train} + \mathbb{A}_S^*$
16: Calculate F1-score$_{secure}$ by testing CNN on $X_{test} + \mathbb{A}_S^*$
17: return $\mathbb{A}_S^*$, F1-score$_{manipulated}$, F1-score$_{secure}$
```

From Line 14 to Line 20, the standard genetic operators selection, crossover, mutation, and clone are used to generate $population$ in the genetic algorithm. On Line 14, the selection operator conducts a weighted sampling without replacement where the weights are proportional to the fitness function values for the current $population$. The selection function in Algorithm 4 randomly samples the current population to return the selected candidates and the remaining candidates as the offspring and parents respectively. On Line 17, the crossover operation is applied between the odd children and even children in the offspring. The crossover function in Algorithm 5 randomly slices and swaps the pixels of the current children. The starting and ending indices for slicing are also selected randomly. On Line 19, random mutations are applied to the pixels of each offspring. The mutation function in Algorithm 6 applies a mask of randomintegers that are added to the pixel values in the current child. Since the masking allows for pixel values in the range of -255 and +255, any mutated pixel values crossing 255 and 0 are taken to be 255 and 0 respectively. The pixels for mutation are selected with a uniform probability.

The new population for the next iteration is cloned on Line 20. Line 21 calls Algorithm 3 to recompute the fitness function values for new $\alpha_{population}$. Line 4 of Algorithm 3 in turn calls an evaluation function to compute the performance metrics on the data subject to adversarial manipulation. These metrics are calculated subject to the current softmax probabilities of the learner. Line 6 of the Algorithm 3 calls Equation (7) to compute the cost of α^*, $cost(\alpha)$.

In Algorithm 2, Line 26 and Line 27 find α^* that is the final converged attack for the adversary. On Line 28, the training algorithm Algorithm 2 returns the final testing performance F1-score on input testing data X_{test} subject to adversarial data manipulation $X_{test} + \alpha^*$.

Algorithm 2. Two-Player Game with Genetic Algorithm

```
1: function twoplayergame-ga ($X_{train}$)
2:   Train CNN on $X_{train}$
3:   $maxpayoff = 0$, exitloop = False
4:   $population = \alpha_{population}$ ▷ Initialize population to size $\psi$
5:   $F(X_{train}) = fitness(X_{train}, \alpha_{population}))$
6:
7:   while gen < maxiter ∧ ¬ exitloop do
8:     $\alpha_{curr}, currpayoff = max(F(X_{train}))$
9:     Train CNN on $X_{train} + \alpha_{curr}$
10:
11:    if abs(currpayoff - maxpayoff) $<$ 0.0001 then
12:      Begin
13:        $maxpayoff = currpayoff$
14:        parents, offspring = selection(population, 0.5)
15:
16:        for child1 in odd offspring and child2 in even offspring do
17:          child1, child2 = clone(crossover(child1,child2))
18:        for mutant in offspring do
19:          mutant = mutation(mutant)
20:        population = clone(parents + offspring)
21:        $F(X_{train}) = fitness(X_{train}, \alpha_{population})$
22:      End
23:    else
24:      exitloop = True
25:  end while
26:  $\alpha_{curr}, maxpayoff = max(F(X_{train}))$
27:  $\alpha^* = \alpha_{curr}$
28:  return $\alpha^*$
29: end function
```

Algorithm 3. Genetic Operators: Fitness Function

```
1: function fitness ($X, Y, \alpha_{population}$)
2:   for $\alpha \in \alpha_{population}$ do
3:     Begin
4:       $metrics = evaluate(X + \alpha, Y)$ ▷ Compute performance measures on manipulated data
5:       $error = \lambda * metrics['recall']$
6:       $\alpha_{fitness} = 1 + error - cost(\alpha)$ ▷ Update fitness values for population
7:     End
8:   return $\alpha_{population}$
9: end function
```

Algorithm 4. Genetic Operators: Selection Function

```
1: function selection ($P, \zeta$)
2:   Retrieve fitness values $W_P$ for all $P$
3:   Sample $P$ without replacement biased by $W_P$ for $\zeta$ percentage of children $C$
4:   return $P - C, C$
5: end function
```

Algorithm 5. Genetic Operators: Crossover Function

```
1: function crossover ($c1, c2$)
2:   Randomly slice $c1$ and $c2$ into $c1_{sliced}$ and $c2_{sliced}$ with minimum width $\eta$
3:   Swap $c1_{sliced}$ with $c2_{sliced}$
4:   return $c1, c2$
5: end function
```

Algorithm 6. Genetic Operators: Mutation Function

```
1: function mutation ($m$)
2:   Randomly select a $step$ between $\delta$ and $-\delta$
3:   Randomly generate a boolean tensor $mask$ for $m$
4:   $m[mask] = m[mask] + step$ ▷ Slice m by mask and update by step
5:   return $m$
6: end function
```

5.2 Simulated Annealing Algorithm

Algorithm 7 gives the adversarial learning algorithm for attacks from adversaries using a simulated annealing algorithm. The game is played on labelled training data X_{train}. Algorithm 8 uses the annealing operator used to generate candidate solutions called α. The game converges onto a final solution α^*.

Algorithm 7 starts the game on Line 2 and ends the game on Line 43. The game iteration is defined in terms of the payoff function value $currpayoff$ which in turn is determined by the convergence criteria for the simulated annealing. The game is initialized between Line 2 and Line 7. On Line 2, a target CNN is trained on X_{train} and tested on X_{test}. The purpose of the game is to find a α^* such that $X_{test} + \alpha^*$ decreases the performance of this CNN found on X_{test}. On Line 3, two flagging variables $maxpayoff$ and $exitloop$ are created to control the convergence of the game. Line 4 initializes the variables controlling the convergence criteria for simulated annealing. Line 6 initializes the candidate solutions to be generated by simulated annealing. Line 6 initializes the mask to be used in the anneal operators. The current mask $mask$ is taken to be all the nonzero pixels contained in the mean images of both the positive class μ_+ and negative class μ_-. Line 7 computes the fitness function value $evalc$ for the initial solution α_c. The game's iteration is given between Line 9 and Line 41. The current temperature for simulated annealing is initialized on Line 4 and is updated on Line 34. On Line 11, the adversary's payoff in the game is taken to be the same as fitness function value for candidate solution α_g. For both the genetic algorithm and the simulated annealing algorithm, the fitness function is computed by the function defined in Algorithm 3 discussed earlier. The game is continued only when there is an increase seen in the payoff $currpayoff$ for the adversary on Line 13. The game ends and the program control shifts to Line 41 if there is no increase in $currpayoff$.

Algorithm 7. Two-Player Game with Simulated Annealing Algorithm

```
1: function twoplayergame-sa ($X_{train}$)
2:   Train CNN on $X_{train}$
3:   $maxpayoff = 0$, exitloop = False
4:   $T_{max} = 1000, T_{min} = 5, \nu = 50, \rho = 0.6, T_{curr} = T_{max}$
5:   Randomly initialize $\alpha_c, \alpha_g, \alpha_n$ to same tensor values
6:   $mask = \mu_+ \wedge \mu_-$   ▷ Initialize the SA parameters and masks
7:   $evalc = fitness(X_{train}, \alpha_c)$
8:
9:   while ¬ exitloop do
10:    $evalg = fitness(X_{train}, \alpha_g)$
11:    $currpayoff = evalg$
12:
13:    if abs(currpayoff - maxpayoff) $<$ 0.0001 then
14:      Begin
15:        $maxpayoff = currpayoff$
16:
17:        while $T_{curr} \geq T_{min}$ do
18:          $i = 1$
19:          while $i \leq \nu$ do
20:            $\alpha_n$ = anneal($\alpha_c, mask$)
21:            $evaln = fitness(X_{train}, \alpha_n)$
22:
23:            if evaln $>$ evalc then
24:              Begin
25:                $\alpha_c = \alpha_n, evalc = evaln$
26:                if evalg $<$ evaln then
27:                  $\alpha_g = \alpha_n, evalg = evaln$
28:              End
29:            else
30:              if random(0,1) $<=$ $e^{(evaln - evalc)/T_{curr}}$ then
31:                $\alpha_c = \alpha_n, evalc = evaln$
32:            $i+ = 1$
33:          end while
34:          $T_{curr}* = \rho$
35:        end while
36:        $\alpha_c = \alpha_g$
37:      End
38:    else
39:      exitloop = True
40:  end while
41:  $\alpha^* = \alpha_g$
42:  return $\alpha^*$
43: end function
```

Within each iteration of the game, the candidate solutions are generated by the annealing operator between Line 17 and Line 35. For every temperature T_{curr}, ν number of candidate solutions α_n are generated. Initially α_n is generated from a randomly initialized α_c. Further solutions from simulated annealing are generated by the current α_c updated on Line 25 and Line 31 according to the searching criteria of the simulated annealing. α_c is also updated on Line 36 at the end of every iteration of the game. The combination of the search parameters sample size ν, reduction rate ρ, maximum temperature T_{max} and minimum temperature T_{min} determines the number and rate of generation of candidate solutions α_n. On Line 20, each α_n is obtained by applying the annealing operator $anneal$ on the best solution α_c defined in Algorithm 8. The masking process used in the operator allows for the randomized selection of tensor regions in the game's search procedure. The random masks are generated by the Boolean tensor mask $maskb$ on Line 2 and the sliced index mask $mask_{sliced}$ on Line 6 alongwith the initialization mask $maska$ passed as a function argument on Line 1 of Algorithm 8. A step of tensor values between parameter $-\delta$ and δ is then applied on the region selected for adversarial manipulation.

Algorithm 8. Simulated Annealing Operators: Anneal Function

```
1:  function anneal ($\alpha, maska$)
2:      Randomly generate a boolean tensor $maskb$ for $\alpha$
3:      $mask = maska \wedge maskb$
4:      Randomly generate a tensor $step$ with values between $\delta$
        and $-\delta$
5:                                   ▷ $step$ has same shape as $\alpha$
6:      Randomly slice $mask$ into $mask_{sliced}$
7:      $\alpha[mask_{sliced}] += step[mask_{sliced}]$
8:                     ▷ Slice $\alpha$ by $mask_{sliced}$ and update by $step$
9:      return $\alpha$
10: end function
```

On Line 21, the fitness function value $evaln$ is computed for every α_n. If a given candidate solution α_n is found to be fitter than the best solution α_c, then the best solution α_c is updated on Line 25. If an increase in adversary's payoff is seen on the given candidate solution α_n, then the adversarial manipulation α_g is updated on Line 27. The corresponding fitness function values $evalc$ and $evalg$ are also simultaneously updated. On Line 30, the search procedure in simulated annealing is randomly restarted with a probability $e^{(evaln-evalc)/T_{curr}}$ depending on the current fitness function and current temperature values. The values of α_c and $evalc$ are updated by the current candidate solutions α_n and $evaln$ at every such restart on Line 31. The current temperature T_{curr} is decreased on Line 34. The best solution α_c found from the simulated annealing iteration within the game iteration is updated on Line 36. At the end of the game, the α_g giving maximum payoff to the adversary is assigned to α^* on Line 41. Line 42 finds the manipulated performance F1-score of the CNN on adversarial data $X_{test} + \alpha^*$ which was trained in Line 2 on the original data X_{train}.

6 Experiments

In this section we discuss the experimental validation and stochastic parameters of the adversarial learning algorithm. During the game, an adversary finds adversarial data manipulations using either a genetic algorithm or a simulated annealing algorithm as the search algorithm. For a two-player game, various parameter settings produce adversarial manipulation α^* on the images such that the positive class examples are misclassified as negative class examples by the CNN aka learner. For example, the CNN misclassifies the handwritten digit 7 which had been positively labelled before adversarial manipulation as the negatively labelled handwritten digit 9 after adversarial manipulation. The CNN is then secured against attacks in a stochastic game with multiple adversaries by defending against adversarial manipulations in many sequential games with two adversaries. The performance of the proposed secure CNN model is also compared with the performance of a CNN model augumented by the data produced from various Generative Adversarial Network. In both the multiplayer game and the two-player game, we observe that the manipulated learner performance is lower than the original learner performance. Also, the secure learner performance is higher than the manipulated learner performance.

TABLE 2
Datasets of Color Images Used in the Experiments

Class Labels	Positive Class	Positive Class Cardinality	Negative Class Cardinality
(2,8)	2	6,990	6,825
(4,9)	4	6,824	6,958
(1,4)	1	7,877	6,824
(5,8)	5	6,313	6,825
(3,8)	3	7,141	6,825
(7,9)	7	7,293	6,958
(6,8)	6	6,876	6,825
(2,6)	2	6,990	6,876

6.1 Dataset Description

We use a cross-validation dataset of colour images split between two class labels. The dataset is taken from the MNIST handwritten images database [31]. The two class labels and their cardinality are described in Table 2. The lower digit is taken to be the positive class. For example, if the class labels are 7 and 9, the class label 7 is taken to be the positive class. The learner is Tensorflow's CIFAR10 CNN model[2] [30]. The learner's testing performance is the baseline performance targeted by the adversary in the game. We assume the adversary has a mean image of positive label data to initialize the population in genetic algorithm Algorithm 2. We also assume that the adversary has a mean image of positive label and negative label data to initialize the masking in simulated annealing algorithm Algorithm 7.

6.2 Genetic Algorithm Validation in Sequential Game

In this section we validate the adversarial data in a sequential game that is constructed by the mutation, crossover, and selection genetic operators defined on the images. The testing performance for mutation, crossover, selection operators and population size on the data manipulated by final α^* is reported in Fig. 3. The x-axis has variation in the parameters for each genetic operator. The y-axis has the learner F1-score performance for the manipulated data.

In the following, the genetic operators and parameters are described in more detail.

Population Initialization. In the initial $\alpha_{population}$ of the images, the pixel values are randomly initialized around the meanimage of the positive class. The range for random pixel values is between the lower bound of RGB pixel value

2. https://www.tensorflow.org/tutorials/deep_cnn#cifar-10_model

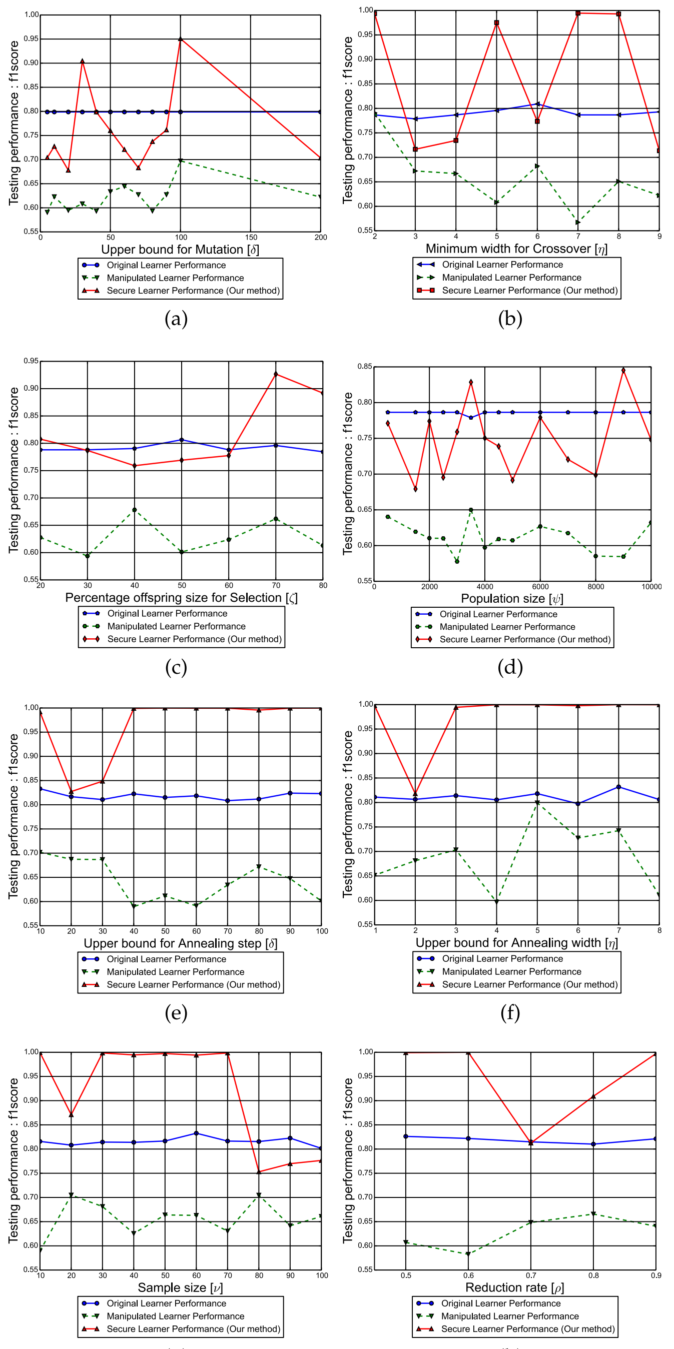


Fig. 3. Testing performance with variations in evolutionary operators consisting of genetic parameters and annealing parameters. Genetic parameters are given in (a), (b), (c), and (d) for mutation step δ, crossover width η, selection size ζ, and population size ψ respectively. Annealing parameters are given in (e), (f), (g), and (h) for annealing step δ, annealing mask width η, annealing sample size ν, and annealing reduction rate ρ respectively. The manipulated learner has lower performance than the original learner. The secure learner has higher performance than the manipulated learner.

(-255) and the upper bound of RGB pixel value (+255). The size of the images is 32*32*3 as required by the CNN model. For the input images manipulated by adding α^*, the pixels with values greater than 255 are set to 255 and pixels with values less than 0 are set to 0.

Mutation Operation. A mask of randomly generated integers between a lower bound -δ (set to -50 by default) and upper bound +δ (set to +50 by default) for the step is added to the current image in the mutation operation. In Fig. 3a, the x-axis is varied by δ—the mutation step. From Fig. 3a we conclude that the F1-score is minimized to 0.5 on the MNIST dataset for δ around 80.

Crossover Operation. For a three-dimensional 32*32*3 RGB image, the height and width indices are randomly selected. The starting index for height is selected between pixels 1 and 16 (half of the largest height). The ending index for height is selected between lower bound η (set to +2 by default) and η (set to +10 by default) from the corresponding starting index of height and upper bound 32. A similar random indexing scheme selects the starting width and ending width of the image. The slice of the starting and ending index of height/width over all the pixels in depth is then swapped between the two images in the crossover operation. In Fig. 3b, the x-axis is varied by η—the crossover width. From Fig. 3b we conclude that the F1-score is minimized to 0.5 on the MNIST dataset for η around 7.

Selection Operation. The selection operation is an extension of random sampling without replacement. The parents for the next generation are randomly chosen from the current generation parents. A ζ (set to 0.5 by default) percentage of the current generation parents are selected to be the offspring for the next generation. The remaining candidates in the current generation of parents are preserved as parents for the next generation. The probability of selecting an offspring is proportional to the fitness values of the current parents. The selected offspring are then changed by crossover and mutation to get the parents for the next generation. Across every generation of the genetic algorithm, the size of the entire population (consisting of current offspring and parents) is fixed to the initial size of the parents. In Fig. 3c, the x-axis is varied by ζ—the selection size. From Fig. 3c we conclude that the F1-score is minimized to 0.5 on the MNIST dataset for ζ around 30 percent.

Population Size. In Fig. 3d, the x-axis is varied by ψ—the population size. ψ is supposed to have an effect on the randomization of the genetic operators. From Fig. 3d, we observe that the testing performance of the manipulated learner decreases by around 20 percent from 0.79 to 0.60 for the F1-score. Then the testing performance of the secure learner trained on the manipulated data increases by around 15 percent from 0.6 to 0.75 for the F1-score. The highest decrease in the manipulated learner's performance is seen at population size 3,000, 8,000 and 9,000. But the highest increase in the performance of the secure learner is seen at 9,000. So we conclude that the secure learner performance increases with an increase in ψ. But the corresponding performance of the manipulated learner is seen to be periodically decreasing with ψ. ψ values of 3,000 and 9,000 seem to be the most suitable for the MNIST dataset.

We find higher values of $\lambda > 1$ are suitable for finding the most suitable genetic algorithm parameters. In all the figures we have used $\lambda = 10$ to minimize the F1-score and maximize the error term in the fitness function of Equation (5). For various settings of the genetic parameters, the game quickly converges into an α^* at Nash equilibrium in a maximum of 20 generations of the genetic algorithm. Randomization and the corresponding effect on the manipulated learner performance increases with attack strengths, iterations and population size in the game.

6.3 Simulated Annealing Validation in Sequential Game

In this section, we validate the adversarial data that is constructed by the annealing operator defined on the images. The testing performance for the annealing operator over steps, masks and its parameters settings on sample size, and reduction rate are reported in Fig. 3. The x-axis shows variation in each parameter's values. The y-axis shows the learner F1-score performance for the original data and the manipulated data. In all the figures, we observe that the secure learner performance is higher than the manipulated learner performance and the manipulated learner performance is lower than the original learner performance.

In the following, the annealing operator and parameters are described in more detail.

Annealing Step. The annealing step parameter δ limits the upper and lower bounds for the pixel values added to the region of the image selected by the annealing operator. δ takes a default value of 20 and is varied between 5 and 100. The pixel values are randomly selected to be any integer value between $-\delta$ and δ. In Fig. 3e, the x-axis is varied by δ. From Fig. 3e we conclude that a δ between 40 and 60 is most suitable for adversarial manipulation. Compared to the original F1-score performance, the drop in manipulated F1-score performance by varying δ varies from 15 to 30 percent whereas the gain in retrained F1-score performance varies from 15 to 50 percent. The annealing step parameter δ is similar to the mutation step parameter δ.

Annealing Mask. The annealing mask width parameter η limits the upper and lower bounds on the indices of the images that have been selected by the annealing operator for adversarial manipulation. η takes a default value of 2 for the lower bound and 10 for the upper bound for an image that is 32 pixels wide and 32 pixels high. The starting and ending indices for annealing are taken to be any random integer between the bounds set by η. A square region with the same starting and ending indices determined by η is then selected for adversarial manipulation. To generate appropriate candidate solutions, an additional condition is that the starting index for η is in the first half of the image and the selected square region has an area of at least 1 pixel. The final mask applied to generate candidate solutions is determined by the randomization resulting from η alongwith the nonzero pixel values found in positive and negative data. In Fig. 3f, the x-axis is varied by the difference in the upper bound and lower bound for η. From Fig. 3f we conclude that a width of 4 is suitable for adversarial manipulation on the given training data. A maximum performance drop and gain of 20 and 50 percent is also observed in Fig. 3f. The annealing mask width parameter η is similar to the crossover width parameter η.

Annealing Sample Size. The annealing sample size parameter ν determines the number of candidate solutions generated in each run of simulated annealing within each iteration of the game. ν takes a default value of 50. In Fig. 3g, ν is varied between 10 and 100. From Fig. 3g we conclude that a sample size of 40 is suitable for adversarial manipulation. We also observe that there is a periodic trend in the manipulated performance across ν. A maximum performance drop and gain of 20 and 50 percent is also observed in Fig. 3g.

Annealing Reduction Rate. The annealing sample size parameter ρ determines the rate of convergence of the simulated annealing runs within each iteration of the game. ρ takes a default value of 0.6. In Fig. 3h, ρ is varied between 0.5 and 0.9. A value of $\rho < 0.5$ ensures that the simulated annealing converges quickly whereas $\rho > 0.5$ ensures it converges slowly. From Fig. 3h we conclude that a value of ρ between 0.5 and 0.6 seems best suited for adversarial manipulation. A maximum performance drop and gain of 25 and 60 percent is also observed in Fig. 3h.

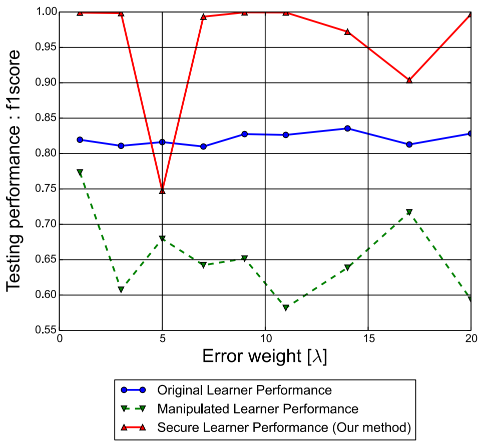


Fig. 4. Testing performance with variation in error weight λ. The manipulated learner has lower performance than the original learner. The secure learner has higher performance than the manipulated learner.

6.4 Fitness Function Validation in Sequential Game

The adversarial manipulation α^* is found on convergence of the game. The game convergence criteria are subject to the fitness function Equation (5) used in the evolutionary algorithm. In this section, we report that by using an α^*, an adversary is able to affect the testing performance of the learner across various parameter settings in the evolutionary algorithm and various combinations of positive and negative classes in the input MNIST database. t-statistics are calculated on the testing F1-score performance to check the effect of α^* across various classes and attack scenarios used in both the genetic algorithm and the simulated annealing algorithm.

For a converged α^* output by the game in either Algorithms 2 or 7, F1-score measures the performance of our algorithm in terms of the testing data X_{test}, manipulated testing data $X_{test} + \alpha^*$ that are seen when the learner is trained on input data X_{train} and manipulated training data $X_{train} + \alpha^*$.

Fitness Function Error Weight. The error weight parameter λ determines the importance of the CNN error with respect to the cost of adversarial manipulation as simulated annealing generates candidate solutions within each iteration of the game. In Fig. 4, λ is varied between 0.1 and 20. A value of $\lambda < 1$ ensures that simulated annealing gives less weight to CNN error rather than game cost to determine the best candidate for adversarial manipulation. A value of $\lambda > 1$ gives more weight to CNN error rather than game cost. A value of $\lambda = 1$ gives the same weight to CNN error and game cost. λ is empirically set for each labelled input. From Fig. 4 we conclude that a value of $\lambda > 1$ is best suited for the current input. Specifically, λ between 10 and 15 seems best suited for the game. A maximum performance drop and gain of 20 and 40 percent is also observed in Fig. 4. Tables 3 and 4 shows the p-values for the learner's F1-score before and after the game. Here t-statistic is an unpaired 2-

TABLE 3
Genetic Algorithm: p-Values Comparison Before and After Game by Varying Parameters for Each Genetic Operator

Genetic Parameter	Model performances t-statistics in Two-player games			
	$CNN_{original}$ *vs* $CNN_{manipulated}$	$CNN_{original}$ *vs* CNN_{secure}	$CNN_{manipulated}$ *vs* CNN_{secure}	Friedman test
Upper bound for Mutation (δ)	8.0×10^{-16}	0.1395	2.5×10^{-5}	3.6×10^{-5}
Minimum width for Crossover (η)	0.0001	0.1446	0.0020	0.0098
Percentage offspring size for Selection (ζ)	4.6×10^{-7}	0.2393	0.0001	0.0111
Population size (ψ)	9.3×10^{-22}	0.00936	7.1×10^{-10}	5.7×10^{-06}

t-statistics are computed between pairs of learner F1-score performance, manipulated learner F1-score performance and secure learner F1-score performance.

sample t-test statistic. We conclude the following from the p-values on the performance of original learner $CNN_{original}$, manipulated learner $CNN_{manipulated-cnn}$ or $CNN_{manipulated}$ and secure learner CNN_{secure}.

- The manipulated learner $CNN_{manipulated}$ under attack has lower performance than the original learner $CNN_{original}$.
- The secure learner CNN_{secure} retrained from the game has higher performance than the manipulated learner $CNN_{manipulated}$ under attack.
- The secure learner CNN_{secure} has equal or higher performance than the original learner $CNN_{original}$.
- The low p-values in Tables 3 and 4 allow us to reject the null hypothesis that the means of the performance measures are the same before and after adversarial data manipulations.
- From the low p-values (<0.05) of the t-statistic comparing original learner $CNN_{original}$ with manipulated learner $CNN_{manipulated}$, the t-statistic comparing manipulated learner $CNN_{manipulated}$ with secure learner CNN_{secure} and the Friedman test statistic in the paired test statistics of Table 3 we conclude that adversarial manipulations from the genetic algorithm affect learner performance with and without adversarial training data.
- From the low p-values (<0.05) of the t-statistic comparing original learner $CNN_{original}$ with manipulated learner $CNN_{manipulated}$, t-statistic comparing original learner $CNN_{original}$ with secure learner CNN_{secure}, the t-statistic comparing manipulated learner $CNN_{manipulated}$ with secure learner CNN_{secure} and the Friedman test statistic in paired test statistics of Table 4 we conclude that adversarial manipulations from simulated annealing algorithm not only affect learner performance with and without adversarial training data but also that the retrained learner has better performance and is comparable to the original learner.
- From the high p-values (>0.05) of t-statistic comparing original learner $CNN_{original}$ with secure learner CNN_{secure} in Table 3 and the low p-values (<0.05) of the t-statistic comparing original learner $CNN_{original}$ with secure learner CNN_{secure} in Table 4 we conclude that the learner with simulated annealing attack scenarios is more robust than the learner with genetic algorithm attack scenarios in the two-player two-label sequential game.
- From the low p-values across Tables 3 and 4, we conclude that our game is not sensitive to the choice of an evolutionary algorithm. The statistical significance of our model generalizes across various parameter randomizations comparing the performances of original learner $CNN_{original}$, manipulated learner $CNN_{manipulated}$ and secure learner CNN_{secure}. In all the attack scenarios and two-label classification problems, we conclude that secure learner CNN_{secure} is robust to adversarial attacks proposed and simulated on the original learner $CNN_{original}$.

6.5 Evolutionary Operations Validation in Stochastic Game

In this section, we report the proposed model performances across various game types. Table 5 reports the performances for a sequential game and Table 6 reports the performances for a stochastic game. The sequential game is a special case of the stochastic game. The F1-scores in the tables are given for original learner $CNN_{original}$, manipulated learner $CNN_{manipulated}$ that is either a CNN based learner $CNN_{manipulated-cnn}$ or a GAN based learner $CNN_{manipulated-gan}$ on original and generated data respectively, secure learner CNN_{secure}. For the purposes of experimentation in a stochastic game, we assume five independent adversaries attack the learner. We also assume the original learner for baseline performance is a CNN model. The CNN is trained/tested on both the original data in the MNIST database and the generated data is obtained from an adversarial network. The lower

TABLE 4
Simulated Annealing Algorithm: p-Values Comparison Before and After Game by Varying Parameters for Each Annealing Operator

Annealing Parameter	Model performances t-statistics in Two-player games			
	$CNN_{original}$ *vs* $CNN_{manipulated}$	$CNN_{original}$ *vs* CNN_{secure}	$CNN_{manipulated}$ *vs* CNN_{secure}	Friedman test
Upper bound for Perturbation step (δ)	1.8×10^{-10}	2.1×10^{-6}	1.85×10^{-10}	4.6×10^{-5}
Upper bound for Perturbation index (η)	2.0×10^{-4}	4.6×10^{-6}	5.4×10^{-7}	3.4×10^{-4}
Reduction rate (ρ)	1.6×10^{-6}	1.0×10^{-2}	4.9×10^{-5}	1.5×10^{-2}
Sample size (ν)	6.8×10^{-11}	1.1×10^{-2}	1.37×10^{-6}	3.7×10^{-4}
Error weight (λ)	5.1×10^{-7}	2.0×10^{-4}	1.9×10^{-7}	3.0×10^{-4}

t-statistics are computed between pairs of learner F1-score performance, manipulated learner F1-score performance and secure learner F1-score performance.

TABLE 5
Two-Player Two-Label Sequential Games Performance Evaluation—F1-Score Before and After Two-Player Game Across Various Combinations of Handwritten Digits

F1-score: Genetic Operators in Two-player Game							F1-score: Annealing Operators in Two-player Game							Class Labels
$CNN_{original}$	$CNN_{manipulated}$					CNN_{secure}	$CNN_{original}$	$CNN_{manipulated}$					CNN_{secure}	
	CNN	$DCGAN$	$IWGAN$	$BEGAN$	$InfoGAN$			CNN	$DCGAN$	$IWGAN$	$BEGAN$	$InfoGAN$		
0.8696	0.7023	0.7881	0.6814	0.7737	0.8019	0.909	0.8464	0.692	0.6834	0.6548	0.7483	0.7759	0.9372	(2,8)
0.8193	0.6354	0.7502	0.7219	0.7487	0.8029	0.9186	0.9582	0.7094	0.763	0.7539	0.8957	0.9218	0.9346	(4,9)
0.9678	0.5288	0.6147	0.631	0.631	0.6311	0.8834	0.9816	0.5953	0.693	0.6319	0.8681	0.634	0.9983	(1,4)
0.8889	0.4881	0.7041	0.679	0.6787	0.6758	0.8141	0.9119	0.6395	0.6547	0.6751	0.6786	0.6757	0.9639	(5,8)
0.8971	0.5183	0.6812	0.6484	0.7887	0.7314	0.9226	0.8974	0.5575	0.6433	0.6479	0.8681	0.7316	0.9977	(3,8)
0.7995	0.6546	0.6592	0.7486	0.6919	0.7738	0.9987	0.8177	0.6181	0.7097	0.8011	0.651	0.6481	0.9991	(7,9)
0.9649	0.5813	0.761	0.7197	0.7995	0.7317	0.9555	0.96	0.6172	0.761	0.7595	0.7057	0.7946	0.83	(6,8)
0.9463	0.5702	0.8873	0.8879	0.8985	0.9299	0.9612	0.9609	0.611	0.8581	0.9057	0.9514	0.95	0.989	(2,6)
t-statistics	7.0×10^{-8}	1.2×10^{-4}	3.5×10^{-5}	3.1×10^{-4}	8.0×10^{-4}	Base	t-statistics	8.2×10^{-9}	3.8×10^{-6}	4.4×10^{-5}	3.0×10^{-3}	1.2×10^{-3}	Base	

We observe a consistent decrease in the manipulated learner $CNN_{manipulated}$ performance tested on adversarial data as compared to learner $CNN_{original}$ performance. We also observe a consistent increase in the secure learner CNN_{secure} performance compared to manipulated learner $CNN_{manipulated}$.

digit in the class labels tuple of Tables 5 and 6 is taken to be the positive label for adversarial manipulation. In the tables, we use the models defined in Section 3.2.

To obtain the generated data, we use four GANs, namely, Conditional DCGAN [18], IWGAN [19], BEGAN [20] and InfoGAN [21]. The data is generated for each pair of positive and negative labels. The CNN trained on the original MNIST data participates in the game whose output is the adversarial manipulation on each of the positive labels creating adversarial data. Then, a CNN trained on the generated data is validated against the adversarial data.

The stochastic game consists of multiple adversaries playing attack scenarios in terms of either genetic operators or annealing operators determining the training algorithm in the game. Following the formulation in Equation (8), by retraining the learner on all the adversarial manipulations, we demonstrate an effective learner robust to combined attacks from all the players. In Tables 5 and 6, for both the original data and generated data, we observe that a learner tested on manipulated data $CNN_{manipulated-cnn}, CNN_{manipulated-gan}$ shows a significant decrease in performance compared to the learner $CNN_{original}$ trained and tested on original data.

By validating the adversarial data against the CNN trained on the output of various GANs, we observe that the effectiveness of the attack decreases with the increased robustness of the GANs. A more successful attack with larger error margins is possible when the CNN participating in the game is trained on original data as well as generated data. We also observe that GA attacks are better than SA attacks. Finally, multiplayer game attacks are better than two-player game attacks. After the game's convergence, the learner can be retrained on the manipulated data to find secure learner CNN_{secure} weights that are secure to further adversarial manipulations on both original and generated data.

The last five rows in Tables 5 and 6 report p-values for the t-statistics comparing secure learner performance with manipulated learner performance where the manipulated learner is trained on either the original data or the generated data. The attack scenarios for generating adversarial manipulations are determined by number and type of players participating in the game. The game is simulated over different combinations of the positive and negative class labels. The t-statistics also validate that our method is immune to various adversarial datasets. We demonstrate that the original CNN model as well as the GAN-based CNN model are vulnerable to proposed adversarial manipulations. The effectiveness of adversarial manipulations is demonstrated to generalize from the two-player game to the multiplayer game. More adversarial manipulations can be obtained by defining more adversarial scenarios comparing the original deep network

TABLE 6
Multiplayer Two-Label Stochastic Games Performance Evaluation— F1-Score Before and After Multiplayer Game Across Various Combinations of Handwritten Digits

F1-score: Genetic Operators in Multiplayer Game							F1-score: Annealing Operators in Multiplayer Game							Class Labels
$CNN_{original}$	$CNN_{manipulated}$					CNN_{secure}	$CNN_{original}$	$CNN_{manipulated}$					CNN_{secure}	
	CNN	$DCGAN$	$IWGAN$	$BEGAN$	$InfoGAN$			CNN	$DCGAN$	$IWGAN$	$BEGAN$	$InfoGAN$		
0.8651	0.6316	0.6748	0.707	0.7564	0.7717	0.8715	0.8466	0.6826	0.6572	0.6193	0.7166	0.7894	0.9474	(2,8)
0.8607	0.6027	0.7461	0.7466	0.7974	0.8154	0.829	0.8212	0.7123	0.7238	0.7387	0.775	0.8153	0.9023	(4,9)
0.9745	0.629	0.6034	0.6314	0.636	0.6316	0.833	0.9777	0.5938	0.6288	0.7005	0.7315	0.7332	0.8679	(1,4)
0.8294	0.6718	0.6825	0.6789	0.6871	0.69	0.8491	0.9096	0.5251	0.6452	0.6956	0.6879	0.6871	0.9998	(5,8)
0.8876	0.7455	0.6698	0.6526	0.7788	0.7315	0.9041	0.9007	0.6306	0.6689	0.6506	0.8283	0.7316	0.8583	(3,8)
0.8572	0.6836	0.6824	0.7898	0.7273	0.7799	0.9866	0.8484	0.7176	0.7136	0.7811	0.7905	0.8317	0.9885	(7,9)
0.9602	0.5827	0.7444	0.6992	0.7927	0.7183	0.8528	0.9523	0.5457	0.6824	0.7475	0.7318	0.7433	0.7697	(6,8)
0.925	0.6304	0.8649	0.8902	0.9131	0.91	0.936	0.9449	0.766	0.8332	0.9022	0.9161	0.9295	0.9364	(2,6)
t-statistics	4.5×10^{-7}	1.4×10^{-4}	5.4×10^{-4}	3.9×10^{-3}	3.2×10^{-3}	Base	t-statistics	1.6×10^{-5}	2.9×10^{-5}	6.2×10^{-4}	2.6×10^{-3}	5.2×10^{-3}	Base	

We observe a consistent decrease in the manipulated learner $CNN_{manipulated}$ performance tested on adversarial data as compared to learner $CNN_{original}$ performance. We also observe a consistent increase in the secure learner CNN_{secure} performance compared to manipulated learner $CNN_{manipulated}$.

model with the manipulated deep network model as well as the retrained deep network model. Better search algorithms and optimization conditions would lead to adversarial data manipulations that are effective on both original data distributions as well as generated data distributions.

7 Conclusion and Future Work

We have formulated a maxmin problem for adversarial learning with both two-player sequential games and multiplayer stochastic games over deep learning networks. The experiments demonstrate the correctness and performance of proposed adversarial algorithm. The algorithm converges onto adversarial manipulations affecting testing performance in deep learning networks. This allows us to propose a secure learner that is immune to the adversarial attacks on deep learning. We have shown that our model is significantly more robust than traditional CNN and GAN under adversarial attacks.

By changing the game formulation, we can experiment with adversarial payoff functions over randomized strategy spaces. The attack scenarios over such strategy spaces would determine multiplayer games over mixed strategies. In the future, we plan to investigate more challenging scenarios where adversaries attack multiple labels simultaneously.

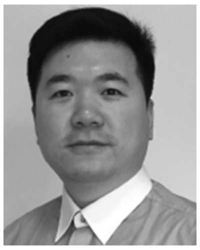
Acknowledgments

This research is funded by The Capital Markets Cooperative Research Centre, Australia.

References

[1] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proc. ACM Symp. Inf. Comput. Commun. Security*, 2006, pp. 16–25.
[2] B. Biggio, G. Fumera, I. Pillai, and F. Roli, "A survey and experimental evaluation of image spam filtering techniques," *Pattern Recognit. Lett.*, vol. 32, no. 10, pp. 1436–1446, Jul. 2011.
[3] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proc. 4th ACM Workshop Security Artif. Intell.*, 2011, pp. 43–58.
[4] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 4, pp. 984–996, Apr. 2014.
[5] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Trans. Signal Inf. Process.*, vol. 3, E2, 2012, doi: 10.1017/atsip.2013.9.
[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley , S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[7] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments," *Int. J. Mach. Learn. Cybern.*, vol. 1, no. 1–4, pp. 27–41, 2010.
[8] A. Kołcz and C. H. Teo, "Feature weighting for improved classifier robustness," in *Proc. 6th Conf. Email Anti-Spam (CEAS)*, Mountain View, CA, USA, 2009.
[9] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Mach. Learn.*, Jun. 2012, pp. 1807–1814.
[10] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, "Support vector machines under adversarial label contamination," *Neurocomput.*, vol. 160, pp. 53–62, 2015.
[11] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. Berkay Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," in *Proc. ACM Asia Conf. Comput. Commun. Security*, 2017, pp. 506–519.
[12] S. Gu and L. Rigazio, "Towards deep neural network architectures robust to adversarial examples," *arXiv preprint arXiv:1412.5068*, 2014, http: //arxiv.org/abs/1412.5068
[13] A. Globerson and S. Roweis, "Nightmare at test time: Robust learning by feature deletion," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 353–360.
[14] B. I. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft, "Learning in a large function space: Privacy-preserving mechanisms for SVM learning," *J. Privacy Confidentiality*, vol. 4, no. 1, 2009, Art. no. 4.
[15] A. Barth, B. I. Rubinstein, M. Sundararajan, J. C. Mitchell, D. Song, and P. L. Bartlett, "A learning-based approach to reactive security," in *Proc. Int. Conf. Financial Cryptography Data Security*, 2010, pp. 192–206.
[16] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: John Wiley & Sons, 2012.
[17] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," in *Proc. 3rd IEEE Eur. Symp. Sec. Privacy*, London, UK, 2016.
[18] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2016.
[19] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein GANs," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5767–5777.
[20] D. Berthelot, T. Schumm, and L. Metz, "BEGAN: boundary equilibrium generative adversarial networks," *arXiv preprint arXiv:1703.10717*, 2017.
[21] X. Chen, X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2172–2180.
[22] M. Kantarcıoğlu, B. Xi, and C. Clifton, "Classifier evaluation and attribute selection against active adversaries," *Data Mining Knowl. Discovery*, vol. 22, no. 1–2, pp. 291–335, 2011.
[23] W. Liu and S. Chawla, "Mining adversarial patterns via regularized loss minimization," *Mach. Learn.*, vol. 81, no. 1, pp. 69–83, 2010.
[24] W. Liu, S. Chawla, J. Bailey, C. Leckie, and K. Ramamohanarao, "An efficient adversarial learning strategy for constructing robust classification boundaries," *Australasian Joint Conf. Artif. Intell.*, 2012, pp. 649–660.
[25] Z. Yin, F. Wang, W. Liu, and S. Chawla, "Sparse feature attacks in adversarial learning," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 6, pp. 1164–1177, Jun. 2018.
[26] J. Zhang, Z.-H. Zhan, Y. Lin, N. Chen, Y.-J. Gong, J.-H. Zhong, H. S. Chung, Y. Li, and Y.-H. Shi, "Evolutionary computation meets machine learning: A survey," *IEEE Comput. Intell. Mag.*, vol. 6, no. 4, pp. 68–75, Nov. 2011.
[27] T. Weise, *Global Optimization Algorithms-Theory and Application*. Second : Self-Published, 2009. University of Kassel, Distributed Systems Group. Copyright (c) 2006-2009 Thomas Weise, licensed under GNU FDL, http://www.it-weise.de/
[28] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm Evol. Comput.*, vol. 1, no. 3, pp. 111–128, 2011.
[29] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 6, no. 5, pp. 443–462, Oct. 2002.
[30] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[31] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database," 1998. [Online]. Available: http://yann. lecun. com/exdb/mnist

Aneesh Sreevallabh Chivukula received the MS degree (by Research) in computer science and engineering from the International Institute of Information Technology(IIIT), Hyderabad, India. He is a PhD student with the Advanced Analytics Institute, the University of Technology Sydney, Australia. He has been working on computational data science driven product development at startup companies and research labs.



Wei Liu received the PhD degree in computer science from the University of Sydney. He is the Data Science Program Leader and a Senior Lecturer with the Advanced Analytics Institute, the University of Technology Sydney, Australia. His research interests are in data mining, machine learning, and artificial intelligence. He has been publishing in the areas of adversarial learning, tensor factorization, causal inference, time series prediction, and outlier detection. He is a member of the IEEE.