# Adversarial Learning Games with Deep Learning Models

Aneesh Sreevallabh Chivukula and Wei Liu
Advanced Analytics Institute, University of Technology Sydney, Australia
AneeshSrivallabh.Chivukula@student.uts.edu.au, Wei.Liu@uts.edu.au

*Abstract*—**Deep learning has been found to be vulnerable to changes in the data distribution. This means that inputs that have an imperceptibly and immeasurably small difference from training data correspond to a completely different class label in deep learning. Thus an existing deep learning network like a Convolutional Neural Network (CNN) is vulnerable to adversarial examples. We design an adversarial learning algorithm for supervised learning in general and CNNs in particular. Adversarial examples are generated by a game theoretic formulation on the performance of deep learning. In the game, the interaction between an intelligent adversary and deep learning model is a two-person sequential noncooperative Stackelberg game with stochastic payoff functions. The Stackelberg game is solved by the Nash equilibrium which is a pair of strategies (learner weights and genetic operations) from which there is no incentive for either learner or adversary to deviate. The algorithm performance is evaluated under different strategy spaces on MNIST handwritten digits data. We show that the Nash equilibrium leads to solutions robust to subsequent adversarial data manipulations. Results suggest that game theory and stochastic optimization algorithms can be used to study performance vulnerabilities in deep learning models.**

**Keywords : Supervised learning, Data mining and knowledge discovery, Evolutionary learning, Adversarial learning, Deep learning, Genetic algorithms, Game theory**

## 1. Introduction

Machine learning algorithms make the stationarity assumption that the training data (used to learn mathematical patterns) and the testing data (evaluating algorithm performance) are sampled from the same underlying probability distribution of independent and identically distributed random variables. However non-stationary data is being increasingly found in real world applications. Designing robust computing systems and machine learning algorithms for such applications is the goal of adversarial learning. It incorporates defence mechanisms into the machine learning algorithm design and action.

Adversarial learning is simulated by training a learning algorithm under various attack scenarios formulated by an intelligent adversary [1]. The optimal attack policy for the adversary is defined in terms of an objective function. Search and optimization algorithms are used to arrive at a satisfactory solution for the objective function. The various adversarial learning algorithms differ in assumptions regarding adversary's knowledge, security violation, attack strategies, attack influence [2].

In this paper we propose a new adversarial learning algorithm for studying deep learning models under attack. Deep learning refers to a class of machine learning algorithms with many stages of nonlinear information processing in hierarchical architectures exploited for pattern classification and feature learning [3]. Deep learning has been found to be susceptible to adversarial examples [4]. Such adversarial examples can be crafted by prior knowledge, observation, and experimentation on the network layers and loss functions in the deep learning model.

Our adversarial examples are generated from algorithms in learning-theoretic game theory. 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 or utility function. The key idea in game theory is that of an equilibirum state from which none of the players have any incentive to deviate. Most common equilibirum is the Nash and Stackelberg equilibrium.

Our adversarial algorithm proposes a game between two players - a data miner or learner and an intelligent adversary. The interactions between the learner and adversary are modelled as a two-player sequential Stackelberg zero-sum game. The payoff for each player is designed as objective functions specifying the attack and 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 and evolutionary computing.

Following are the major contributions of this paper.

- We formulate the problem of finding defence mechanisms in adversarial learning as a maxmin optimization problem in learning-theoretic game theory.
- We develop a new algorithm for adversarial learning in deep learning models. We do not assume the adversary knows anything about the deep network structure which is close to real life settings.
- Our algorithm can adapt to continuous adversarial data manipulations unlike most of the existing adversarial learning algorithms. In each iteration of

the adversarial learning algorithm, we define a tensors based fitness function and payoff function to evaluate the solution domain represented by genetic operators.

- The theoretical goal is to determine a manipulating tensor on the input data that finds learner decision boundary where many positive labels become negative labels. Upon convergence the learner can retrain to find weights that are robust to adversarial attacks.

The paper starts with related work in Section 2 comparing the new approach with existing approaches. The pseudocode and experiments for proposed algorithm are presented in Section 3 and Section 4 respectively. The paper ends in Section 5 with a summary of current work and future work.

## 2. Related Work

The existing adversarial learning algorithms are summarized in Table 1, Table 2. The table's columns list the various criteria for comparing adversarial learning algorithms. The table's rows list the various algorithms under comparison. The algorithms are compared on attack strategy, search algorithm, adversary's knowledge. 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 "adversary's knowledge" is the semantic information of the adversary. Our algorithm is termed "Game theory : deep learning".

### 2.1. Adversarial Security Mechanisms

For various attack scenarios in adversarial learning, several defence mechanisms have been proposed for learner security [2, 5, 12, 13]. Such mechanisms include attempts at designing secure learning algorithms [2], multiple classifier systems [5], privacy-preserving machine learning [12] and use of randomization or disinformation to mislead the adversary [13].

Biggio et al. [2] have an empirical framework for evaluation of learner security. This framework extends the model selection and performance evaluation steps of pattern classification [14] by adding system design steps of "security by design" rather than "security by obscurity". The additional steps proposed in the framework are useful for evaluating the security of both generative learning and discriminative learning. Depending on the goal, knowledge and capability of the adversary, these steps are classified in terms of attack influence, security violation and attack specificity.

The attack influence can be causative or exploratory. Causative attack affects both training and testing data. Exploratory attack affects only testing data.

The security violation can target either integrity or availability or privacy of the learner. A machine learning algorithm whose integrity is compromised cannot detect malicious behaviour of the adversary. The integrity of an algorithm with many false negatives is compromised. A machine learning algorithm whose availability is compromised has severely degraded performance for legitimate users. The availability of an algorithm with many false positives is compromised. The privacy of an algorithm whose detailed feedback is made public is compromised.

The attack specificity can be either targeted or indiscriminate for attacks that influence prediction or action of the algorithm. In targeted attacks the attack is directed at only a few instances of the training or testing data. In indiscriminate attacks the attack is directed at an entire class of instances or objects.

Our algorithm has causative attack influence, integrity security violation, targeted attack specificity.

### 2.2. Adversarial Networks

In deep learning networks, adversarial examples are generated by applying a small but intentional worst-case perturbation to examples in the cross-validation data. Such perturbed input results in an incorrect output with high confidence. Goodfellow et al. [4] state that the primary cause of deep learning networks vulnerability to adversarial examples is their linear nature in high dimensional search spaces. Another observation is that deep learning networks do not perform well on points in search space that do not have high probability in the training data distribution. This means that inputs that have an imperceptibly and immeasurably small difference from training data correspond to a completely different class label in deep learning. Thus with increasing dimensionality of the input data, a deep learning network has a greater vulnerability to adversarial examples.

Thus Goodfellow et al. argue for the need of having an adversarial training procedure with the objective to minimize the worst case error when the data is perturbed by an adversary. Adversarial learning is proposed to be a form of active learning where the deep learning network is able to request labels on new points and learn better regularization through adversarial training. In active learning process, the labels on faraway points are supplied by a domain expert whereas the labels on nearby points are supplied by a heuristic labeler.

Adversarial samples are also known to transfer between deep learning models. Papernot et al. [9] have a practical demonstration of this phenomenon. The adversarial examples are constructed to control the integrity of a target Deep Neural Network (DNN). The adversary has no access to the target DNN's architecture, parameters, training data.

Gu and Rigazio [10] study robustness of DNNs by studying pre-processing and training strategies accounting for structure of adversarial examples and model's network topology. The pre-processing is done with Denoising Autoencoders (DAEs).

Instead of an adversarial network, we have a genetic algorithm constructing perturbations to training data.

| Adversarial algorithm | Attack strategy | Search algorithm |
|---|---|---|
| Classifier ensembles [5] | Reorder features by importance for discriminant function | Randomized sampling |
| Feature weighting [6] | Addition/deletion of binary features | Feature bagging |
| SVM : inputs [7] | Train noise injection | Gradient ascent |
| SVM : labels [8] | Label noise injection | Gradient ascent |
| Deep learning [4] | Linear perturbation on x | Backpropagation with L-BFGS |
| Adversarial networks : DNN [9] | Observe DNN outputs given inputs chosen by the adversary | Jacobian-based dataset augmentation |
| Adversarial networks : DAE [10] | Gaussian additive noise | Stacking DAEs into a feed forward neural network |
| Game theory : support vector machines [11] | Delete different features from different data points | Quadratic programming |
| **Game theory : deep learning (Our method)** | Move positive samples towards negative samples | Genetic algorithm |

TABLE 1: Adversarial Algorithms Comparision

| Adversarial algorithm | Adversary's knowledge | Learning games |
|---|---|---|
| Classifier ensembles [5] | Train features | None |
| Feature weighting [6] | Train features | None |
| SVM : inputs [7] | Gradient of loss | None |
| SVM : labels [8] | Train labels | None |
| Deep learning [4] | Train and test data | None |
| Adversarial networks : DNN [9] | Test data | None |
| Adversarial networks : DAE [10] | Test data | None |
| Game theory : support vector machines [11] | Train features | Non-zero sum game |
| **Game theory : deep learning (Our method)** | Train and test data | Constant sum game |

TABLE 2: Adversarial Algorithms Comparison

## 2.3. Learning-Theoretic Game Theory

Game theory has been adapted to machine learning algorithms vulnerable to adversarial data manipulations [15]. A game is assumed to mimic the parameter tuning actions in the learning algorithm. Moreover, the data miner is assumed to use only one learning algorithm throughout the game. This is because adjusting parameters in existing model is assumed to be computationally less expensive than building a new model. The adversary is assumed to modify learning strategy to avoid detection from learner. At the same time, the learner updates its model based on new threats from the adversary. The equilibrium is reached when neither the learner nor the adversary has incentive to play the game. The learner has no incentive to play the game that leads to too many false positive with too little increase in true positives. The adversary has no incentive to play the game that increases utility of false negatives not detected by the learning algorithm. A game ends with payoffs to each player based on their objectives and actions. All player's are assumed to act in their rational interest to maximize the payoffs. This assumption at every stage of game, eliminates Nash equilibria with non-credible threats and creates an equilibrium called the subgame perfect equilibrium. Here perfect equilibrium means each player knows about the others utility function. The players utility functions vary by application domain.

Globerson and Roweis [11] discuss a classification algorithm with a game theoretic formulation. The proposed algorithm is robust to features deletion according to a minmax objective function optimized by quadratic programming. In Liu and Chawla [16], 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 proposed minmax problem for optimization is solved without making assumptions on the data distribution underlying training and testing data. Each player's move is based on the observation of the opponent's last play. The adversary iteratively attacks the data miner by best possible strategy for transforming the original training data. The data miner reacts by rebuilding classifier based on data miner's observations of the adversarys modifications to the training data. The adversarys strategy of play is determined independently by the adversary. The game is repeated until adversary's payoff does not increase or the maximum number of iterations is reached. Liu et al. [17] propose an extension to Liu and Chawla where one-step game is used to reduce computing time of the minmax algorithm. The one-step method converges to Nash equilibrium by utilizing Singular Value Decomposition (SVD).

For deriving the payoff functions in the game, we assume that the adversary has no knowledge of either the network layers or loss functions in the deep learning model.

## 2.4. Stochastic Optimization

Evolutionary Computing (EC) has been used to generate rule-based data mining models and search algorithms with attribute interactions [18]. The EC 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 [19], [20].

We use EC algorithm to generate adversarial examples. It is a genetic algorithm with standard selection, crossover, mutation operators generating adversarial examples. By using EDA algorithms, the genetic operators can define explicit probabilistic distributions on the candidate solutions in multivariate models [21].

## 3. Algorithm

In this section we discuss the problem formulation and pseudocode for the proposed adversarial learning algorithm.

### 3.1. 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 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. A 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 $w$, the best strategy for the leader is

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

Similarly for L's move $\alpha$, 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 classifier are assumed to sum to a constant profit $\Phi$. 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 have trained 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 CNN layers for the input $\alpha$ 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 adversary is the leader L and learner is the follower F.

Using a genetic algorithm, the adversary searches for data manipulations that maximize classification error $error(w)$. The fitness function in genetic algorithm $J_L(\alpha, w)$ is defined to be the adversary's payoff function. The genetic algorithm converges when adversary does not see an increase in payoff function or the maximum number of iterations are reached. The game and genetic algorithm are assumed to have same convergence criteria.

For input training data $X_{train}$ and corresponding labels $Y_{train}$, the adversary searches for a move $\alpha$ that maximizes following payoff function or fitness function $J_L(\alpha)$ where $error$ is the classification error as measured by $recall$ for current adversarial data. The term $tensornorm$ is a modified $\ell_2$ norm for the current $\alpha$.

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

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

$$tensornorm(\alpha) = \sqrt{\Sigma\alpha^2/(32 * 32 * 3)}/255 \tag{7}$$

$tensornorm(\alpha)$ term in $J_L(\alpha)$ is affected by the initialization of $\alpha$ population whereas the $error$ term is affected by the genetic operators. $tensornorm(\alpha)$ ensures that the adversary makes a minimum of changes to the current $\alpha$ while maximizing the error or minimizing the corresponding recall. $tensornorm(\alpha)$ is enhanced by an weighting term $\lambda$ whose default value 10. $\lambda$ is empirically evaluated for each dataset. A constant 1 is then added to $J_L(\alpha)$ to ensure a positive fitness function in the genetic algorithm.

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 have assumed that the adversary tries to reduce the true positives in classifier performance regardless of the changes to true negatives. By assuming the error to depend on recall, the adversary attempts to convert true positives to either false positives or false negatives. The recall is inturn determined by the learner's performance.

### 3.2. Game Illustration

We choose a Convolutional Neural Network (CNN) as the learner. CNN architecture's input layers and output layer are described in [22] and available in the Tensorflow API [1] as the CIFAR10 model. The CNN has input layers consisting of convolution layers, maxpooling layers, regularization layers, 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.

Figure 1 has an illustration of the game. The $CNN_{initial}$ is trained on training data ($X_{train}$) and evaluated on testing data $X_{test}$ to give a performance metric $recall(X_{test}) \mid CNN(X_{train})$ given as "learner performance" for method 1 in Figure 1. The adversary targets this performance by engaging the CNN in a game. The game has steps executed by the adversary and learner for each interaction. In these steps, the adversary is targeting the learner by data produced from genetic operators. The learner is then adapting for the adversarial data by retraining the CNN. Upon convergence, the game outputs a adversarial data manipulation step $\alpha$ that is added to the original data sample to create final adversarial data sample.

1. https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/index.html

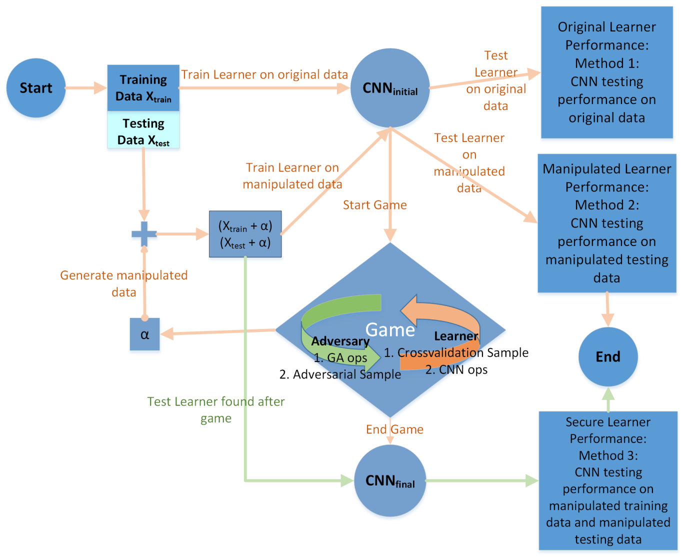

Start
Training Data $X_{train}$
Testing Data $X_{test}$
Train Learner on original data
$CNN_{initial}$
Test Learner on original data
Original Learner Performance: Method 1: CNN testing performance on original data
Train Learner on manipulated data
Test Learner on manipulated data
Start Game
Manipulated Learner Performance: Method 2: CNN testing performance on manipulated testing data
$(X_{train} + \alpha)$ $(X_{test} + \alpha)$
Generate manipulated data
α
Game
Adversary
1. GA ops
2. Adversarial Sample
Learner
1. Crossvalidation Sample
2. CNN ops
End
Test Learner found after game
End Game
$CNN_{final}$
Secure Learner Performance: Method 3: CNN testing performance on manipulated training data and manipulated testing data

Figure 1: A flow chart illustrating the benefits of a game theoretic learner. The game has Adversary and Learner as the players. The game produces a final deep learning network $CNN_{final}$ that is better equipped to deal with the adversarial manipulations than the initial deep learning network $CNN_{initial}$.

The CNN trained and tested on the adversarial data sample has performance $recall(X_{test}+\alpha) \mid CNN(X_{train}+\alpha)$ given as "manipulated learner performance" for method 2 in Figure 1. This recall is significantly less than the original CNN recall $recall(X_{test}) \mid CNN(X_{train})$ on original training and testing data $(X_{train}, X_{test})$. A new Convolutional Neural Network $CNN_{final}$ is then retrained to adapt on the manipulated data $(X_{train}+\alpha)$ and $(X_{test}+\alpha)$. $CNN_{final}$ performance on manipulated data is $recall(X_{test} + \alpha) \mid CNN(X_{train} + \alpha)$ given as "secure learner performance" for method 3 in Figure 1. Method 3 is our proposed method. It is found to be better than the manipulated CNN recall $recall(X_{test} + \alpha) \mid CNN(X_{train} + \alpha)$.

Thus we arrive at the conclusion that the new $CNN_{final}$ has successfully adapted to adversarial data manipulations as compared to the original $CNN_{initial}$. Our algorithm is able to find a data sample that affects the training of a CNN. The CNN that is able to recover from our adversarial attack is better equipped to deal with unforseen changes in the data. The game between adversary and learner allows us to produce a CNN that is able to estimate changes in the underlying data distribution.

## 3.3. Game Algorithm

Algorithm 1 gives the training algorithm that takes into consideration adversarial attacks. As input, Algorithm 1 requires the training data $X_{train}$ labelled by $Y_{train}$ and testing data $X_{test}$ labelled by $Y_{test}$. In our algorithm, each example in the input data is a three dimensional tensor of RGB pixel values. Algorithm 2, Algorithm 3, Algorithm 4, Algorithm 5 give the genetic operators used by Algorithm 1 to search for candidate solutions.

The Algorithm 1 initializes the game by training the CNN on Line 3 to store the weights to disk. The fitness function values are computed on Line 6 for each randomly initialized $\alpha$. $\alpha$ belongs to a genetic population $\alpha_{population}$ operating on the input data $X_{train}$ and $Y_{train}$. $\alpha_{population}$ is randomly initialized around the mean of the positive class. It is assigned to $population$ on Line 5. A variable $maxpayoff$ is used to keep track of the adversary's current payoff in current iteration of the game from Line 8 to Line 30. The if condition on Line 12 ensures that the algorithm converges when $maxpayoff$ does not exceed current payoff $currpayoff$ by a small number 0.0001. In each game iteration, the current $\alpha_{curr}$ giving best fitness value is selected on Line 9. On Line 10, the CNN is retrained to react to attack $X_{train}+\alpha_{curr}$. The variable $maxpayoff$ is updated on Line 14 if $\alpha_{curr}$ satisfies the game convergence criteria.

From Line 15 to Line 25, the standard genetic operators selection,crossover,mutation,clone are used to generate $population$ in the genetic algorithm.

On Line 15, selection operator does a weighted sampling without replacement where weights are proportional to fitness function values for current $population$. Selection function in Algorithm 3 randomly samples the current population to return selected candidates and remaining candidates as the offspring and parents respectively. On Line 19, crossover operation is applied between the odd children and even children in the offspring. Crossover function in Algorithm 4 randomly slices and swaps the pixels of current children. The starting and ending indices for slicing are also selected randomly. On Line 23, random mutations are applied to pixels of each offspring. Mutation function in Algorithm 5 applies a mask of randomintegers that are added to the pixel values in 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 uniformly probability.

The new population for next iteration is cloned on Line 25. Line 26 calls Algorithm 2 to recompute the fitness function values for new $\alpha_{population}$. Line 4 of Algorithm 2 inturn calls an evaluation function to compute the performance metrics on data subject to adversarial manipulation. These metrics are calculated subject to the current softmax probabilities of the learner. Line 6 of the Algorithm 2 calls Equation 7 to compute tensornorm of $\alpha^*$, $tensornorm(\alpha)$.

In Algorithm 1, Line 31 and Line 32 find $\alpha^*$ that is the final converged attack for the adversary. On Line 35, the training algorithm Algorithm 1 returns the final testing performance $f1score$ on input testing data $X_{test}$ and $Y_{test}$ subject to adversarial data manipulation $X_{test} + \alpha^*$.

**Algorithm 1** Training Game

```
Training Input:
 1: $X_{train}, Y_{train}, X_{test}, Y_{test}$
Training Output:
 2: $\alpha^*$
 3: train($X_{train}, Y_{train}$)
 4: $maxpayoff = 0$, exitloop = False
 5: $population = \alpha_{population}$        ▷ Initialize population to size $\psi$
 6: $F(X_{train}) = fitness(X_{train}, Y_{train}, \alpha_{population})$
 7:
 8: while gen<maxiter $\wedge \neg$ exitloop do
 9:     $\alpha_{curr}, currpayoff = max(F(X_{train}))$
10:     cnn_train($X_{train} + \alpha_{curr}, Y_{train}$)
11:
12:     If abs(currpayoff - maxpayoff) $< 0.0001$ then
13:         Begin
14:             $maxpayoff = currpayoff$
15:             parents,offspring = selection(population,0.5)
16:
17:             for child1 in odd offspring and child2 in even offspring do
18:                 Begin
19:                     child1, child2 = clone(crossover(child1,child2))
20:                 End
21:             for mutant in offspring do
22:                 Begin
23:                     mutant = mutation(mutant)
24:                 End
25:             population = clone(parents + offspring)
26:             $F(X_{train}) = fitness(X_{train}, Y_{train}, \alpha_{population})$
27:         End
28:     else
29:         exitloop = True
30: end while
31: $\alpha_{curr}, maxpayoff = max(F(X_{train}))$
32: $\alpha^* = \alpha_{curr}$
33: $cnn\_train(X_{train}, Y_{train})$
34: $f1score = cnn\_test(X_{test} + \alpha^*, Y_{test})$
35: return $\alpha^*, f1score$
```

**Algorithm 2** 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:             $\alpha_{fitness} = 1 + error - tensornorm(\alpha)$        ▷ Update fitness values for population
7:         End
8:     return $\alpha_{population}$
9: end function
```

**Algorithm 3** 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 4** 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 5** 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
```

## 4. Experiments

In this section we discuss the experimental validation and parameters of the adversarial learning algorithm.

### 4.1. Dataset description

We use a crossvalidation dataset of colour images split between two class labels. The dataset is taken from the MNIST handwritten images database [23]. The two class labels and their cardinality are described in Table 3. 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 model [2]. The learner's architecture is described in [22]. 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 for initializing the $\alpha_{population}$ in the genetic algorithm Algorithm 1.

### 4.2. Illustrative Examples

Figure 2 has examples for 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, changing shape and size of the image. In Figure 2, Figure (a) has handwritten digit 2 manipulated to looks like 8 by adding pixels. Figure (b) has handwritten digit 3 manipulated to looks like 8 by changing thickness and shape. Figure (c) has handwritten digit 4 manipulated to looks like 9 by deleting pixels and changing shape. Figure (d) has handwritten digit 7 manipulated to looks like 9 by deleting pixels and changing shape.

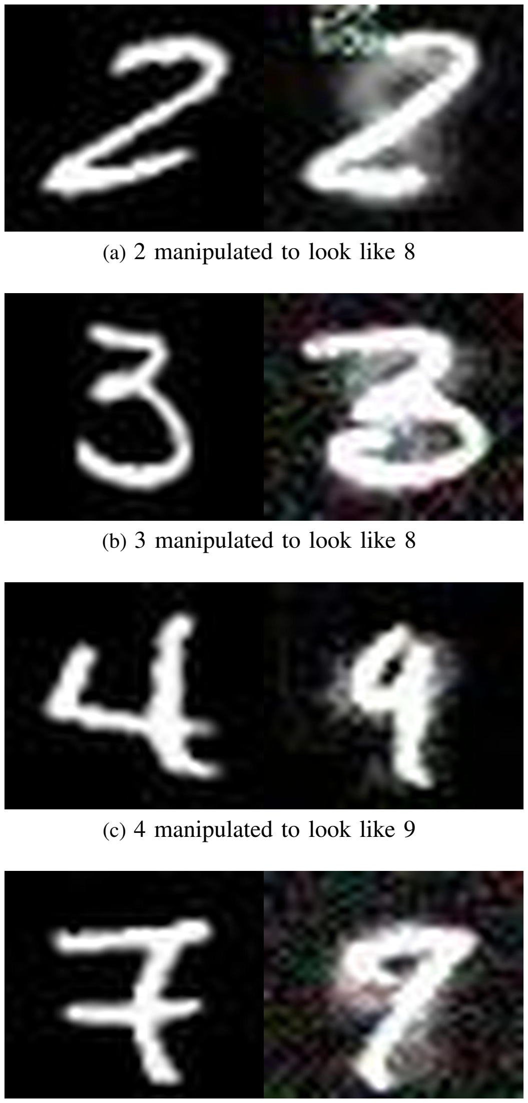
### 4.3. Search algorithm validation

During the game, an adversary finds adversarial examples using a genetic algorithm as the search algorithm. In this section, we report the effect of genetic operators in the search algorithm. We conclude that an adversary is able to affect the testing performance of the CNN learner by varying the search algorithm parameters. These parameters allow us to produce a adversarial manipulation $\alpha^*$ on the images such that the positive class examples are misclassified as negative class examples by the CNN. For example, the CNN misclassifies the handwritten digit 7 that is positively

2. https://github.com/tensorflow/tensorflow/tree/master/tensorflow/models/image/cifar10

| Class Labels | Positive Class | Positive Class Cardinality | Negative Class Cardinality |
|---|---|---|---|
| (2,8) | 2 | 6990 | 6825 |
| (4,9) | 4 | 6824 | 6958 |
| (1,4) | 1 | 7877 | 6824 |
| (5,8) | 5 | 6313 | 6825 |
| (3,8) | 3 | 7141 | 6825 |
| (7,9) | 7 | 7293 | 6958 |
| (6,8) | 6 | 6876 | 6825 |
| (2,6) | 2 | 6990 | 6876 |

TABLE 3: Datasets of colour images used in the experiments


(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

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

labelled before adversarial manipulation as the negatively labelled handwritten digit 9 after adversarial manipulation.

The adversarial data is constructed by the mutation, crossover, selection genetic operators defined on images. The testing performance for mutation, crossover, selection operators and population size on data manipulated by $\alpha^*$ is reported in Figure 3, Figure 4, Figure 5 and Figure 6 respectively. The x-axis is has variation in the parameters for each genetic operator. The y-axis has the learner f1score performance for manipulated data. In all the figures, we observe that the secure learner performance is higher than the manipulated learner performance. From the figures, we can also find the best values of the genetic parameters that minimize f1score to around 0.5.

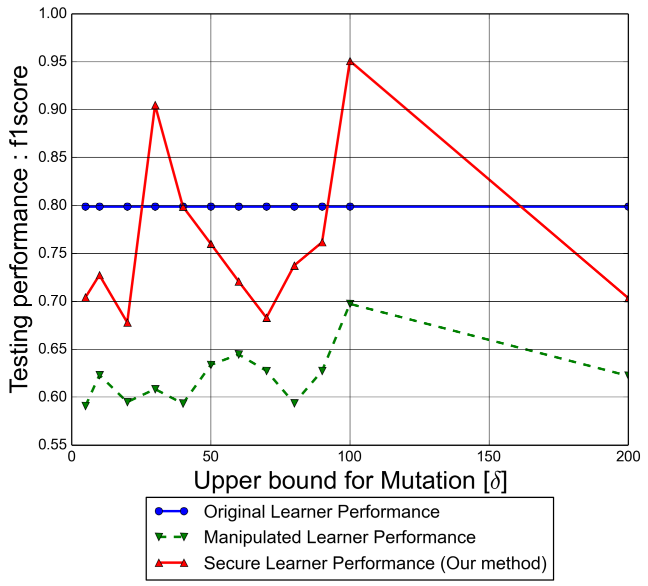


Figure 3: Testing performance with variation in mutation step $\delta$. Secure learner has higher performance than manipulated learner.

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

**Population initialization** In the initial $\alpha_{population}$ of images, the pixel values are randomly initialized around the meanimage of the positive class. The range for random pixel values is between lower bound of RGB pixel value(-255) and upper bound of RGB pixel value(+255). The size of images is 32*32*3 as required by the CNN model. For input images manipulated by adding $\alpha^*$, 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

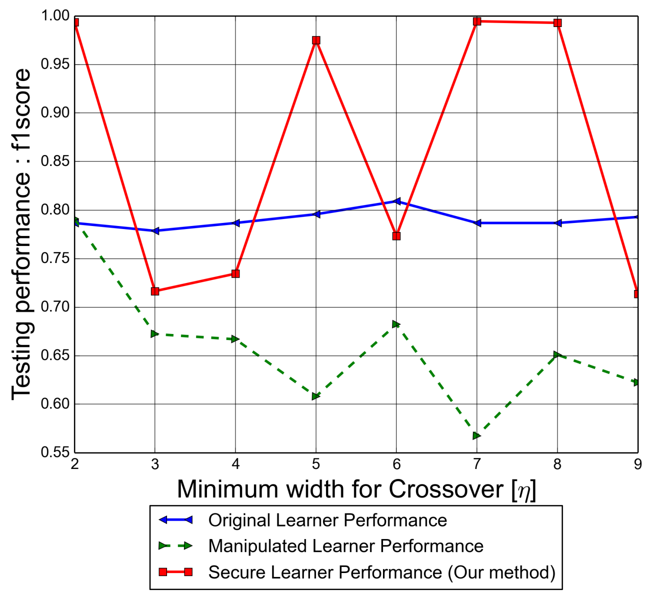


Figure 4: Testing performance with variation in crossover width $\eta$. Secure learner has higher performance than manipulated learner.

integers between a lower bound -$\delta$(set to -50 by default) and upper bound +$\delta$(set to +50 by default) for the step is added to the current image in mutation operation. In Figure 3, the x-axis is varied by $\delta$ - the mutation step. From Figure 3 we conclude that the f1score is minimized to 0.5 on MNIST dataset for $\delta$ 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 largest height). The ending index for height is selected between lower bound $\eta$(set to +2 by default) and $\eta$(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 starting and ending index of height/width over all the pixels in depth is then swapped between the two images in crossover operation. In Figure 4, the x-axis is varied by $\eta$ - the crossover width. From Figure 4 we conclude that the f1score is minimized to 0.5 on MNIST dataset for $\eta$ around 7.

**Selection operation** Selection operation is an extension of random sampling without replacement. The parents for next generation are randomly chosen from current generation parents. A $\zeta$(set to 0.5 by default) percentage of the current generation parents are selected to be the offspring for next generation. Remaining candidates in current generation of parents are preserved as parents for next generation. The probability of selection of 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 next generation. Across every generation of the genetic algorithm, the size of entire population(consisting of current offspring and parents) is fixed to the initial size

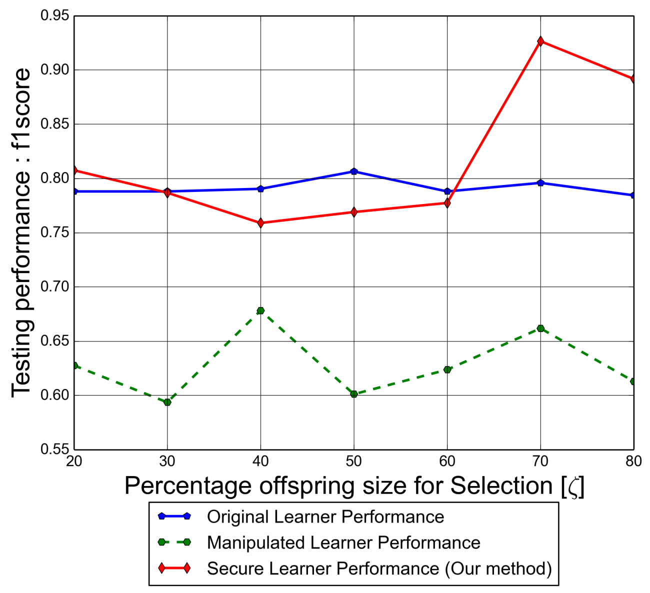


Figure 5: Testing performance with variation in selection size $\zeta$. Secure learner has higher performance than manipulated learner.

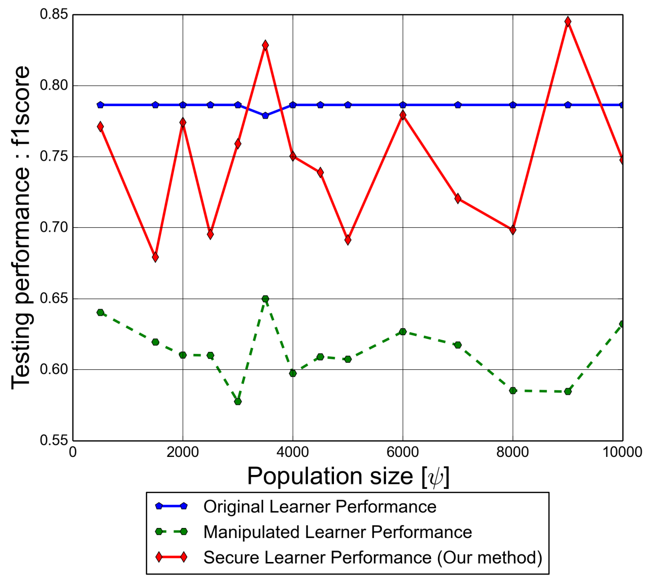


Figure 6: Testing performance with variation in population size $\psi$. Secure learner has higher performance than manipulated learner.

of the parents. In Figure 5, the x-axis is varied by $\zeta$ - the selection size. From Figure 5 we conclude that the f1score is minimized to 0.5 on MNIST dataset for $\zeta$ around 30 %.

**Population size** In Figure 6, the x-axis is varied by $\psi$ - the population size. $\psi$ is supposed to have an effect on the randomization of the genetic operators. From Figure 6, we observe that the testing performance of the manipulated learner decreases by around 20% from 0.79 to 0.60 f1score. Then the testing performance of the secure learner trained on the manipulated data increases by around 15% from 0.6

to 0.75 f1score. The highest decrease in the manipulated learner's performance is seen at population size 3000, 8000 and 9000. But the highest increase in performance of secure learner is seen at 9000. So we conclude that the secure learner performance increases with an increase in $\psi$. But the corresponding performance of the manipulated learner performance is seen to be periodically decreasing with $\psi$. $\psi$ values of 3000 and 9000 seem to be 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 $\lambda = 10$ for minimizing the f1score and maximizing the error term in fitness function of Equation 5. For various settings of the genetic parameters, the game quickly converges onto an $\alpha^*$ at Nash equilibrium in a maximum of 20 generations of the genetic algorithm. Randomization and corresponding effect on the manipulated learner performance increases with attack strengths, iterations and population size in the game.

## 4.4. Fitness function validation

The adversarial manipulation $\alpha^*$ is found on convergence of the game. The game convergence criteria are subject to the fitness function Equation 5 used in the genetic algorithm. In this section, we report that using an $\alpha^*$, an adversary is able to affect the testing performance of the learner across various positive and negative classes in the MNIST database. t-statistics are calculated on the testing f1score performance to check the effect of $\alpha^*$ across various classes and attack scenarios.

For a converged $\alpha^*$ output by the game in Algorithm 1, $f1score$ 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 $X_{train} + \alpha^*$.

Table 4 has the testing performance of the learner under various attack scenarios of varying attack strengths of the adversary. The attack strengths are expressed in terms of the genetic parameters. A low strength attack scenario corresponds to $\delta = 50, \eta = 5, \zeta = 30, \psi = 1000$. A high strength attack scenario corresponds to $\delta = 100, \eta = 10, \zeta = 50, \psi = 3000$.

Table 5 shows the p-values seen for learner's $f1score$ before and after the game. Here t-statistic is an unpaired 2-sample t-test statistic. The p-values are reported for the t-statistic between pairs of columns in Table 4. The p-values for friedmantest-statistic are for all columns giving testing performance in Table 4. We conclude the following from the p-values in Table 5.

- The manipulated learner under attack has lower performance than the learner trained on original data.
- The secure learner retrained from the game has higher performance the manipulated learner under attack.
- The secure learner has equal or higher performance than the original learner.
- The low p-values in Table 5 allow us to reject the null hypothesis that the means of the performance measures are the same before and after adversarial data manipulations.
- From low p-values ($<0.05$) of comparison1 t-statistic, comparison3 t-statistic and friedmantest-statistic in paired test statistics we conclude that adversarial manipulations affect learner performance with and without adversarial training data.
- From high p-values ($>0.05$) of comparison2 t-statistic we conclude that secure learner is robust to adversarial attacks simulated on the learner.
- From low p-value ($<0.05$) of comparison2 t-statistic for Population Size ($\psi$) we conclude that further attack scenarios can be obtained by finding suitable $\psi$ across datasets.

Thus, from Table 4 and Table 5 we observe that a learner trained on manipulated data shows a significant decrease in testing performance as compared to the learner trained on original data. After the game's convergence, the learner can retrain on the manipulated data to find weights that are secure to further adversarial manipulations on both training and testing data.

## 5. Conclusion and Future Work

We have presented a maxmin problem for adversarial learning in deep learning networks. The experiments on images data demonstrate the correctness and performance of proposed algorithm. The algorithm converges onto an adversarial dataset affecting testing performance in deep learning. This allows us to propose a secure learner that is immune to the adversarial attacks on deep learning.

An efficient search algorithm and fitness function would generate multilabel adversarial data for training deep learning networks. Dimensionality in multilabel data can be tackled by training the algorithm on multiple processing units. By changing the deep learning architecture, we can also experiment with various classification functions. By changing the genetic operators in the fitness function evaluation, we can experiment with various evolutionary computing and multi-objective optimization methods for stochastic optimization.

## 6. Acknowledgements

This research is funded by the Capital Markets Cooperative Research Centre.

## References

[1] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. ACM, 2011, pp. 43–58.

[2] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack," *IEEE transactions*

| f1score: Low Strength Attack Scenario | | | f1score: High Strength Attack Scenario | | | Class Labels |
|---|---|---|---|---|---|---|
| Learner | Manipulated Learner | Secure Learner (Our method) | Learner | Manipulated Learner | Secure Learner (Our method) | |
| 0.8421 | 0.6788 | 0.7785 | 0.859 | 0.7868 | 0.84 | (2,8) |
| 0.7915 | 0.6552 | 0.7807 | 0.7253 | 0.6786 | 0.7938 | (4,9) |
| 0.9736 | 0.6648 | 0.9813 | 0.9767 | 0.5634 | 0.9886 | (1,4) |
| 0.8295 | 0.6366 | 0.8391 | 0.8371 | 0.6499 | 0.8354 | (5,8) |
| 0.8629 | 0.6812 | 0.9624 | 0.857 | 0.7172 | 0.8721 | (3,8) |
| 0.8024 | 0.619 | 0.8669 | 0.8025 | 0.6471 | 0.7841 | (7,9) |
| 0.6382 | 0.6257 | 0.8202 | 0.7082 | 0.6168 | 0.9598 | (6,8) |
| 0.876 | 0.826 | 0.9974 | 0.8747 | 0.8087 | 0.9233 | (2,6) |

TABLE 4: (7,9) Genetic Algorithm : Performance Evaluation - f1scores before and after game across various combinations of handwritten digits. We observe a consistent decrease in the manipulated learner performance trained on adversarial data as compared to learner performance trained on original data. We also observe a consistent increase in the secure learner performance compared to manipulated learner.

| Genetric Operator | comparison1 t-statistic | comparison2 t-statistic | comparison3 t-statistic | friedmantest-statistic |
|---|---|---|---|---|
| Upper Bound for Mutation ($\delta$) | 8.02e-16 | 0.1395 | 2.46e-05 | 3.59e-05 |
| Minimum width for Crossover ($\eta$) | 0.0001 | 0.1446 | 0.0020 | 0.0098 |
| Percentage offspring size for Selection ($\zeta$) | 4.60e-07 | 0.2393 | 0.0001 | 0.0111 |
| Population Size ($\psi$) | 9.30e-22 | 0.00936 | 7.05e-10 | 5.69e-06 |

TABLE 5: (7,9) Genetic Algorithm : p-values comparison before and after game by varying parameters for each genetic operator. comparison1 t-statistic is computed between learner f1score performance and manipulated learner f1score performance. comparison2 t-statistic is computed between learner f1score performance and secure learner f1score performance. comparison3 t-statistic is computed between manipulated learner f1score performance and secure learner f1score performance.

| Genetric Operator | comparison1 t-statistic | comparison2 t-statistic | comparison3 t-statistic | friedmantest-statistic |
|---|---|---|---|---|
| Upper Bound for Perturbation Step ($\delta$) | 1.80e-10 | 2.07e-06 | 1.85e-10 | 4.54e-05 |
| Upper Bound for Perturbation Index ($\eta$) | 1.98e-04 | 4.60e-06 | 5.37e-07 | 3.35e-04 |
| Reduction Rate ($\rho$) | 1.60e-06 | 1.01e-02 | 4.93e-05 | 1.50e-02 |
| Sample Size ($\nu$) | 6.78e-11 | 1.06e-02 | 1.37e-06 | 3.71e-04 |
| Error Weight ($\lambda$) | 5.10e-07 | 1.96e-04 | 1.85e-07 | 3.00e-04 |

TABLE 6: (7,9) Simulated Annealing : p-values comparison before and after game by varying parameters for each genetic operator. comparison1 t-statistic is computed between learner f1score performance and manipulated learner f1score performance. comparison2 t-statistic is computed between learner f1score performance and secure learner f1score performance. comparison3 t-statistic is computed between manipulated learner f1score performance and secure learner f1score performance.

| Genetric Operator | comparison1 t-statistic | comparison2 t-statistic | comparison3 t-statistic | friedmantest-statistic |
|---|---|---|---|---|
| Upper Bound for Perturbation Step ($\delta$) | 8.80e-11 | 2.10e-04 | 2.57e-08 | 2.25e-04 |
| Upper Bound for Perturbation Index ($\eta$) | 1.11e-08 | 4.13e-05 | 1.03e-07 | 8.05e-04 |
| Reduction Rate ($\rho$) | 1.72e-03 | 4.46e-02 | 2.45e-03 | 2.24e-02 |
| Sample Size ($\nu$) | 4.61e-13 | 4.18e-07 | 8.15e-11 | 4.54e-05 |
| Error Weight ($\lambda$) | 2.54e-07 | 2.85e-10 | 5.84e-12 | 1.23e-04 |

TABLE 7: (4,9) Simulated Annealing : p-values comparison before and after game by varying parameters for each genetic operator. comparison1 t-statistic is computed between learner f1score performance and manipulated learner f1score performance. comparison2 t-statistic is computed between learner f1score performance and secure learner f1score performance. comparison3 t-statistic is computed between manipulated learner f1score performance and secure learner f1score performance.

*on knowledge and data engineering*, vol. 26, no. 4, pp. 984–996, 2014.

[3] L. Deng, “Three classes of deep learning architectures and their applications: a tutorial survey,” *APSIPA transactions on signal and information processing*, 2012.

[4] 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, pp. 2672–2680.

[5] B. Biggio, G. Fumera, and F. Roli, “Multiple classifier systems for robust classifier design in adversarial environments,” *International Journal of Machine Learning and Cybernetics*, vol. 1, no. 1-4, pp. 27–41, 2010.

[6] A. Kołcz and C. H. Teo, “Feature weighting for improved classifier robustness,” in *CEAS09: sixth conference on email and anti-spam*, 2009.

[7] B. Biggio, B. Nelson, and P. Laskov, “Poisoning attacks against support vector machines,” *29th International Conference on Machine Learning (ICML)*, pp. 1807–1814, Jun. 2012.

[8] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, and F. Roli, “Support vector machines under adversarial label contamination,” *Neurocomputing*, vol. 160, pp. 53–62, 2015.

[9] 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,” *2017 ACM Asia Conference on Computer and Communications Security*, 2016.

[10] S. Gu and L. Rigazio, “Towards deep neural network architectures robust to adversarial examples,” *CoRR*,

vol. abs/1412.5068, 2014.

[11] A. Globerson and S. Roweis, "Nightmare at test time: robust learning by feature deletion," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 353–360.

[12] B. I. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft, "Learning in a large function space: Privacy-preserving mechanisms for svm learning," *Journal of Privacy and Confidentiality*, vol. Vol.4 : Iss.1, Article 4., 2009.

[13] A. Barth, B. I. Rubinstein, M. Sundararajan, J. C. Mitchell, D. Song, and P. L. Bartlett, "A learning-based approach to reactive security," *International Conference on Financial Cryptography and Data Security*, pp. 192–206, 2010.

[14] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.

[15] M. Kantarcıoğlu, B. Xi, and C. Clifton, "Classifier evaluation and attribute selection against active adversaries," *Data Mining and Knowledge Discovery*, vol. 22, no. 1-2, pp. 291–335, 2011.

[16] W. Liu and S. Chawla, "Mining adversarial patterns via regularized loss minimization," *Machine learning*, vol. 81, no. 1, pp. 69–83, 2010.

[17] W. Liu, S. Chawla, J. Bailey, C. Leckie, and K. Ramamohanarao, *An efficient adversarial learning strategy for constructing robust classification boundaries*. Australasian Joint Conference on Artificial Intelligence, 2012, pp. 649–660.

[18] 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 Computational Intelligence Magazine*, vol. 6, no. 4, pp. 68–75, 2011.

[19] T. Weise, *Global optimization algorithms-theory and application*. Citeseer, 2009.

[20] M. Hauschild and M. Pelikan, "An introduction and survey of estimation of distribution algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 3, pp. 111–128, 2011.

[21] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

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

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