Neurocomputing 267 (2017) 572–580

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Sensitivity based robust learning for stacked autoencoder against evasion attack

Patrick P.K. Chan[a], Zhe Lin[a,*], Xian Hu[a], Eric C.C. Tsang[b], Daniel S. Yeung[c]

[a] School of Computer Science and Engineering, South China University of Technology, Guangzhou, China
[b] Faculty of Information Technology, Macau University of Science and Technology, Macau,China
[c] IEEE SMC Society, USA

ARTICLE INFO

Article history:
Received 3 April 2017
Revised 7 June 2017
Accepted 15 June 2017
Available online 5 July 2017

Comunicated by Nianyin Zeng

Keywords:
Deep learning
Adversarial learning
Robustness
Evasion attack
Sensitivity

ABSTRACT

Although deep learning has achieved excellent performance in many applications, some studies indicate that deep learning algorithms are vulnerable in an adversarial environment. A small distortion on a sample leads to misclassification easily. Until now, the vulnerability issue of stacked autoencoder, which is one of the most popular deep learning algorithms, has not been investigated. In this paper, we firstly investigate the existing evasion attack to stacked autoencoder in an effort to understand whether, and to what extent, they can work efficiently. A robust learning algorithm which minimizes both its error and sensitivity is then proposed for stacked autoencoder. The sensitivity is defined as the change of the output due to a small fluctuation on the input. As the proposed algorithm considers not only accuracy but also stability, a more robust stacked autoencoder against evasion attack is expected. The performance of our methods is then evaluated and compared with conventional stacked autoencoder and denoising autoencoder experimentally in terms of accuracy, robustness and time complexity. Moreover, the experimental results also suggest that the proposed learning method is more robust than others when a training set is contaminated.

1. Introduction

Deep learning, which is a powerful machine learning method, has been successfully applied in many applications [1], especially image recognition [2–5], natural language understanding [6,7], and signal recognition [8,9]. Representation learning in deep learning methods is able to discover important feature representation for discrimination from raw data in a classification problem. It reduces the demand on manual design of features. One popular example is the stacked autoencoder [10]. Its learning process contains two phases. The unsupervised pre-training phase aims to learn a representation by reducing the reconstruction error of an autoencoder using unlabeled samples one layer at a time. After all hidden layers are pre-trained, prediction error on a supervised task is then minimized in the supervised fine-tuning phase.

Recently, the security problem of machine learning techniques received increasing attention. Many studies [11,12] have revealed that conventional machine learning techniques are vulnerable to an adversarial environment in which an adversary carefully manipulates input samples to mislead decisions of the system. The accuracies of these conventional techniques may drop significantly due to violation of the underlying assumption of data stationarity (i.e., training and test samples follow the same distribution) [13]. Many studies have investigated the security issue of machine learning in different scenarios [14–16] and applications recently [17–19].

Although deep learning methods have dramatically improved the state-of-the-art methods, many studies show that deep learning methods are sensitive to adversarial attacks, i.e., a sample with small distortion can easily mislead the decision of a deep learning method [20]. Most studies on robust deep learning methods focus on deep conventional neural network. For example, a training set is perturbed by a small transformation to increase the robustness of the network [21,22], and some features are blinded randomly to increase the difficulty of the attack [23,24]. However, to the best of our knowledge, there is no investigation on the robustness of the stacked autoencoder, which is one of the most popular deep learning methods, in an adversarial environment.

In this paper, we firstly discuss whether and how the stacked autoencoder is vulnerable to an adversarial attack. A robust learning algorithm which minimizes not only the error but also the sensitivity measure on training samples is then proposed for stacked

* Corresponding author.
E-mail addresses: patrickchan@ieee.org (P.P.K. Chan), chirplam@foxmail.com (Z. Lin), huxian.scut@gmail.com (X. Hu), cctsang@must.edu.mo (E.C.C. Tsang), danyeung@ieee.org (D.S. Yeung).

http://dx.doi.org/10.1016/j.neucom.2017.06.032
0925-2312/

autoencoder. The sensitivity measure is defined as the change of learner's outputs when the input has a small fluctuation. It has been shown that methods with sensitivity measure achieve good performances in many applications, for example, classifier training [25], dynamic fusion [26] and model selection [27]. The algorithm can be applied to the unsupervised representation learning and the supervised fine-tuning phase for the stacked autoencoder. As our proposed algorithm focuses not only on the accuracy but also on the stability, a more robust stacked autoencoder against the adversarial attack using our proposed methods is expected. The performance of our proposed methods using evasion attack, which aims to decease the classification accuracy of a classifier on test samples by manipulating their features, is evaluated and compared with the conventional stacked autoencoder and the denoising autoencoder experimentally. Moreover, we also show that experimentally, the proposed learning method is robust with a noisy training set.

The rest of the paper is organized as follows. Section 2 introduces the related concepts and techniques used in this study, including deep learning and adversarial learning. Section 3 presents our proposed methods and also explains its underlying concept. The algorithm of Evasion attack used in this paper is introduced in Section 4. The experimental evaluation of our proposed and existing methods is discussed in Section 5. Finally, the conclusion is given in Section 6.

2. Background

Concepts and methods used in this study will be introduced in this section. We will firstly discuss evasion attack, and then introduce the stacked autoencoder.

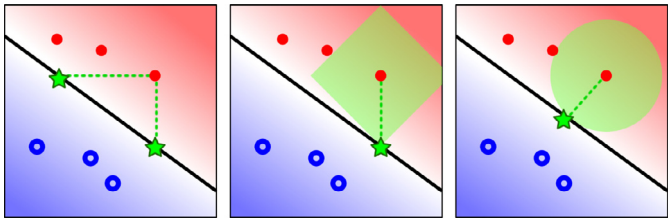
2.1. Adversarial attack

Machine learning methods have been applied to many security related applications in which an adversary aims to downgrade the performance of a system by manipulating the data. In this arm race between the attacker and the defender, the attack side proposes novel attack strategies [28] and explores vulnerabilities of security systems [20], while robust learning algorithms [22] are devised and inputs are carefully selected [23,24] by defenders.

Three different aspects of an adversarial attack against machine learning are considered by the taxonomy proposed in [29,30]: attack influence, security violation, and attack specificity. According to *attack influence*, attack can be separated into poisoning (or causative) and evasion (or exploratory) attack. Poisoning attack [31,32] misleads a system by manipulating training samples while test samples are contaminated in the exploratory attack [13]. *Security violation* can be caused by availability, integrity or privacy attack. Availability attack downgrades the overall performance of a system. Integrity attack only reduces the accuracy on classifying malicious samples. Privacy attack retrieves protected information from a system. *Attack specificity* mentions whether an attack focuses on specific samples (*i.e.,* targeted) or a broad range of samples (*i.e.,* indiscriminate).

The evasion attack discussed in this paper can be regarded as an indiscriminate exploratory integrity attack according to the aforementioned taxonomy. An adversary aims to reduce the classification accuracy of a system by manipulating any test sample in the attack. The optimal evasion [13] aims to generate an attacked sample $\mathbf{x}^* \in \mathscr{X}$ that evades detection by minimally manipulating the original sample $\mathbf{x} \in \mathscr{X}$. The minimal manipulation is measured by a distance function in the feature space. The optimal evasion can be formulated as:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}'} d(\mathbf{x}, \mathbf{x}') \tag{1}$$



(a) L_0 (b) L_1 (c) L_2

Fig. 1. Examples of evasion attacks with L_0, L_1 and L_2 against a linear classifier.

$$\text{s.t. } f(\mathbf{x}) \neq f(\mathbf{x}') \tag{2}$$

where $f : \mathscr{X} \mapsto \mathscr{C}$ is a classifier which outputs the class of x. $d : \mathscr{X} \times \mathscr{X} \mapsto \mathbb{R}$ is the distance function which measures the difference between $\mathbf{x}$ and $\mathbf{x}'$. Common distance functions in evasion attack are L_0 [28], L_1 and L_2 [15], defined as:

$$L_p(\mathbf{x}, \mathbf{x}') = \left(\sum_{i=1}^{m} \left| x_i - x_i' \right|^p \right)^{\frac{1}{p}} \tag{3}$$

where x_i is the ith feature of $\mathbf{x}$, m is the number of features, $\mathbf{x}$ and $\mathbf{x}'$ are the original and attacked samples. When $p = 0$, it is a special case in which L_0 returns the number of features manipulated in an attack sample. Fig. 1 illustrates evasion attacks with L_0, L_1 and L_2 against a linear classifier. The solid circles in red and the hollow circles in blue represent samples in two classes. The straight line denotes the decision plane of the linear classifier. The targeted sample ($\mathbf{x}$) is the solid red circle on the right hand side. Evasion attack misleads the decision of the classifier on $\mathbf{x}$ by manipulating its features according to different distance constraints. The attacked sample ($\mathbf{x}^*$) with minimum manipulation is denoted by a green star. There are two green stars in the evasion attack with L_0 since the sample can mislead the decision by manipulating either one feature.

2.2. Deep learning: autoencoder

Deep neural networks consist of a number of hidden layers. The traditional learning method for neural networks, *i.e.,* gradient descent, does not work well in a neural network with deep architecture due to gradient vanishing and gradient explosion [10]. The principle of deep learning is to train the intermediate hidden layers one at a time. Once a good representation has been obtained, the network can be used as an initial state for training by using supervised gradient based optimization [33,34].

One of the most popular examples of deep learning is stacked autoencoder [10]. In the first training phase, *i.e.,* unsupervised representation learning, each autoencoder takes the output of its previous layer as the input and the reconstruction error is minimized with an optimization algorithm. The objective function is $L(\mathbf{h}, \hat{\mathbf{h}})$, where $\mathbf{h}$ and $\hat{\mathbf{h}}$ denote the original input and the reconstructed input respectively, and L is a loss function which measures the difference between $\mathbf{h}$ and $\hat{\mathbf{h}}$. A regularization term may be added in this objective function. For example, a sparsity term which penalizes non-zero values is added in the sparse autoencoder. The goal of autoencoder is to extract an abstract representation of the data which preserves most information of $\mathbf{x}$.

When a discriminative representation is obtained, the whole network including the output layer is re-trained in the supervised fine-tuning phase. All weights of the network are adjusted in order to minimize the empirical error of the network on training samples measured, which is defined as $L(\mathbf{y}, \hat{\mathbf{y}})$, where $\mathbf{y}$ represents the real original input, and $\hat{\mathbf{y}}$ denotes the output of the network.

In order to propose robust features, a revised stacked autoencoder named as stacked denoising autoencoder (SDA) [35,36] is proposed. By reconstructing the original input from its corrupted ones, SDA is able to learn more robust feature and to tolerate noise. Robust stacked autoencoder (R-SAE) [37] aims to reduce the influence of noisy samples and outliers on training a stacked autoencoder by minimizing the maximum correntropy criterion (MCC). As R-SAE does not consider the performance of the trained stacked autoencoder on an attacked sample, it may be vulnerable under evasion attack.

3. Proposed robust learning method with sensitivity for stacked autoencoder

In this section, our proposed sensitivity-based robust learning method for stacked autoencoder against the evasion attack is introduced. We firstly describe the algorithm and the implementation details on the sensitivity measure. Finally, the explanation on why the proposed model is more robust in comparison with conventional stacked autoencoder and stacked denoising autoencoder is given.

3.1. Algorithm

Conventional learning algorithm of the stacked autoencoder mainly focuses on minimizing the reconstruction error in the representation learning phase and the mean square error in the fine-tuning phase. However, a classifier considering only error may be vulnerable in an adversarial environment [13]. Our proposed learning algorithm aims to reduce not only the error but also the sensitivity of the stacked autoencoder. By reducing the sensitivity of the stacked autoencoder, it will make evasion attacks more difficult since manipulation on a sample causes smaller change on its output in comparison with conventional stacked autoencoder. most of them are generated based on a gradient descent method. As a result, the trained classifier should be accurate and also robust under an adversarial attack.

Given a training set for the unsupervised representation learning, $D_u = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(n_u)}\}$, where n_u is the number of samples in D_u and $\mathbf{x}^{(i)}$ is the ith sample with m feature values, *i.e.,* $\mathbf{x}^{(i)} = [x^{(i,1)}, x^{(i,2)}, \ldots, x^{(i,m)}]^T$. The encoder of a kth layer autoencoder is defined as:

$$\mathbf{h}_k^{(i)} = f_{en}(\mathbf{h}_{k-1}^{(i)}) = a(\mathbf{W}_k^{en}\mathbf{h}_{k-1}^{(i)}), \tag{4}$$

where $\mathbf{h}_{k-1}^{(i)}$ is the input of the kth layer autoencoder, which is one of the outputs of the encoder in the previous layer corresponding to $\mathbf{x}^{(i)}$. For the first layer, $\mathbf{h}_0^{(i)} = \mathbf{x}^{(i)}$. $\mathbf{W}_k^{en}$ is a matrix containing the weights of the encoder of the kth layer autoencoder, and $a()$ denotes an activation function. Then the encoded input will be decoded according to:

$$f_{de}(f_{en}(\mathbf{h}_{k-1}^{(i)})) = a\big(\mathbf{W}_k^{de} f_{en}(\mathbf{h}_{k-1}^{(i)})\big), \tag{5}$$

where $\mathbf{w}_k^{de}$ denotes the matrix containing the weights of decoder of kth layer autoencoder. The objective function of our proposed algorithm for the representation learning is defined as:

$$\min_{\mathbf{W}_k^{en}, \mathbf{W}_k^{de}} \sum_{i=1}^{n_u} \left(||\mathbf{h}_{k-1}^{(i)} - f_{de}(f_{en}(\mathbf{h}_{k-1}^{(i)}))||_2 + \lambda sen_k^{(i)}\right) \tag{6}$$

The first term in (6) represents the reconstruction error (*i.e.,* the difference between the original sample and its reconstruction), while the second one, $sen_k^{(i)}$, is the additional term in our model for the k-layer autoencoder. λ is the tradeoff parameter. $sen_k^{(i)}$ measures the output change of the autoencoder with fluctuated $x^{(i)}$. The sensitivity measure, defined by (7), is the mean value of the squared output of the encoder difference between a sample $\mathbf{x}$ and unseen samples within the Q-neighborhood of $\mathbf{x}$ [27,38].

$$sen_k^{(i)} = E\left(\left|\left|f_{en}(\mathbf{h}_{k-1}^{(i)}) - f_{en}\left(\mathbf{h}_{k-1}^{(i)} + \Delta\mathbf{h}\right)\right|\right|_2\right) \tag{7}$$

where $\Delta\mathbf{h} = \{\Delta h_i | i = 1, 2, \ldots, m\}^T$, and $-q \le ||\Delta h_i|| \le q$. $\mathbf{h}_{k-1}^{(i)} + \Delta\mathbf{h}$ are the unseen samples in Q-neighborhood of $\mathbf{h}_{k-1}^{(i)}$. q controls the size of a Q-neighborhood. A larger region is considered when q is large. When $q = 0$, a Q-neighborhood only contains $x^{(i)}$ and $sen_k^{(i)}$ is equal to 0, *i.e.,* our algorithm is equivalent to the original stacked autoencoder.

Similarly, the sensitivity term can also be considered in the supervised fine-tuning phase. Given a training set with labels, $D_s = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{y}^{(2)}), \ldots, (\mathbf{x}^{(n_s)}, \mathbf{y}^{(n_s)})\}$, where $n_s = |D_s|$. $\mathbf{x}^{(i)}$ is the ith sample while $\mathbf{y}^{(i)}$ represents the label of $\mathbf{x}^{(i)}$. For the 2-class problem, $\mathbf{y}^{(i)}$ contains either -1 or $+1$. $\mathbf{y}^{(i)} = [y^{(i,1)}, y^{(i,1)}, \ldots, y^{(i,c)}]^T$ for c-class problem. If $\mathbf{x}^{(i)}$ belongs to the rth class, for $j \ne r$, $y^{(i,j)} = 0$; otherwise, $y^{(i,j)} = 1$. The objective function of the fine-tuning is defined as:

$$\min \sum_{i=1}^{n_s} (||\mathbf{y}^{(i)} - g(\mathbf{x}^{(i)}))||_2 + \lambda sen^{(i)}) \tag{8}$$

where g is the pretrained stacked autoencoder, and λ is the trade-off parameter. The first and the second components measure the error and the sensitivity, respectively. The definition $sen^{(i)}$ is similar to $sen_k^{(i)}$ defined in (7), but $sen^{(i)}$ considers the input space:

$$sen^{(i)} = E\left(\left|\left|g(\mathbf{x}^{(i)}) - g(\mathbf{x}^{(i)} + \Delta\mathbf{x})\right|\right|_2\right) \tag{9}$$

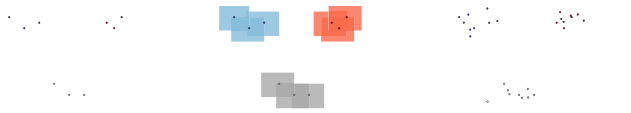
where $\Delta\mathbf{x} = \{\Delta x_i | i = 1, 2, \ldots, m\}^T$, and $-q \le ||\Delta x_i|| \le q$. When $q = 0$, the algorithm is equivalent to the original stacked autoencoder.

Different from the conventional stacked autoencoder, our proposed methods minimize not only the error but also the sensitivity. The sensitivity measure penalizes the fluctuation of the classifier's outputs in Q-neighborhoods in which the size is controlled by q. q provides a geometric interpretation in the input space. As discussed before, the algorithms of our proposed methods neglect the sensitivity term since the Q-neighborhoods are empty when $q = 0$. On the other side, when q increases, larger regions are considered, *i.e.,* the sensitivity of the output is expected to be stable in larger regions. However, it may not be true since, if true distributions of the samples of different classes are complicated, outputs of the classifier should be naturally fluctuated. Any available information on the input data distribution may be useful for determining the size of q.

The sensitivity measure and the generalization ability are usually in conflict with each other, *i.e.,* a stable classifier may be less accurate than a sensitive one [13,14]. Therefore, λ is considered as a trade-off parameter between the error and the sensitivity measure. When λ is small, the training error becomes the dominating factor in the objective function. The trained model aims to maximize its generalization ability and ignore its robustness. On the other hand, the importance of the sensitivity measure increases with the increase of λ. The output of the model should be more robust under evasion attack, i.e. more manipulation is required on a sample for a successful attack. However, the generalization ability of the classifier may not be good.

3.2. Calculation of sensitivity measure

Sensitivity measure, shown in (7) and (9), represents the change of function's outputs when the input is perturbed slightly within the Q-neighborhoods. An artificial dataset containing training samples of three classes, and their Q-neighborhoods are shown in Fig. 2(a) and (b), respectively.



(a) Original data (b) Region of Q-neighborhoods (c) Artificial samples in Q-neighborhoods

Fig. 2. Example of Q-neighborhoods.

(a) A decision boundary considering only the error term (b) A decision boundary considering the error and the sensitivity term

Fig. 3. A decision boundary (a) considering only the error term, and (b) considering only the error term and the sensitivity measure on an artificial dataset with three classes.

$sen_k^{(i)}$ and $sen^{(i)}$ can be approximated using Monte Carlo method [39]. To be more specific, we assume that $\Delta\mathbf{h}$ and $\Delta\mathbf{x}$ follow a Gaussian distribution with mean $\mu = 0$. The standard deviation of the distribution is controlled by q which is the size of the Q-neighborhood. Therefore, $\Delta\mathbf{h} \sim N_m(0,\ q^2)$ and $\Delta\mathbf{x} \sim N_m(0,\ q^2)$. A number of samples are then generated and used to calculate $sen_k^{(i)}$ and $sen^{(i)}$. Fig. 3 shows the original dataset with 10 additional samples generated by Monte Carlo method for each training sample.

According to the Monte Carlo method, a closer estimation is obtained when more points are sampled. However, more samples also yield higher computational cost. The number of samples should be carefully determined in consideration of the nature of the application and the performance issue.

3.3. Why does our method work?

As discussed in Section 2.1, evasion attack [13,31] aims to mislead the decision of a classifier on a sample with the minimum manipulation. The optimal (or sub-optimal) attacked sample is crafted according to the derivative of the classification function, *i.e.,* change of the output. A feature which, with a small change of its value, affects the output of the classifier significantly, is more preferable in an evasion attack. Our proposed methods consider not only the error but also the sensitivity measure. The output values around the training samples of a classifier trained with our algorithm should be more stable than the traditional classifier, *i.e.,* the change of the classifier's output is more gentle in our methods. Therefore, the influence on the output of our method caused by an attacked sample is relatively smaller than the one of the traditional methods. Hence the cost of a successful evasion increases.

On the other hand, the effectiveness of our model can be discussed from a geometrical point of view. Fig. 3(a) shows a decision boundary which only considers the error function on an artificial dataset with three classes. As the only objective of learning is to minimize the error, some samples may be close to the decision plane, such that, small manipulation may be able to evade the detection. On the contrary, for the decision plane trained by minimizing the error function and also the sensitivity term, as shown in Fig. 3(b), the output fluctuation in the Q-neighborhood is relative small. As a result, the decision boundary will be located further away from the samples, *i.e.,* large Δx is required for a successful evasion attack.

4. Evasion attack for multi-class classification problem

The algorithm of the evasion attack for a multi-class classification problem used in this study is discussed in this section. For a classification problem with c number of classes, c classifiers ($f_i(\mathbf{x})$, where $i = 1, 2, \ldots, c$) are trained to estimate the confidence of a sample $\mathbf{x}$ on classes. $\mathbf{x}$ is classified as the class j if $f_j(\mathbf{x}) > f_i(\mathbf{x})$ where, $i \neq j$.

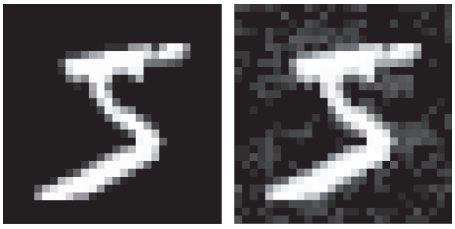
The evasion attack strategy used in this study aims to reduce the accuracy of a classifier on test samples by manipulating their features. The worst scenario of evasion attack in which an adversary has complete knowledge on the targeted system is considered. Evasion attack is implemented using the optimal attack strategy, which evades detection with minimum manipulation. As suggested by Russu et al. [40], the evasion attack with L_2 is harder for detection than the one with L_1 and L_0 since pixels in the image are changed only slightly. L_2 is applied to measure the difference between an original and the attacked sample in this study. The solution of the optimization problem (1) for a differentiable classification function is obtained by the gradient descent, as shown in Algorithm 1, approximately. The algorithm of an evasion attack reduces the estimated confidence of $\mathbf{x}$ (*i.e.,* $f_y(\mathbf{x})$) on its true class (y) iteratively by the gradient descent method. The iterative process stops when $\mathbf{x}$ is evaded successful, or the maximum number of iteration or the manipulation reaches their upper bounds, respectively.

Algorithm 1 Evasion attack.

Input: $\mathbf{x}$: the original sample; y: the class label of $\mathbf{x}$; $f_i(\mathbf{x})$: a classifier function outputs confidence of sample belongs i class; *max_iter*: maximum number of loop; *max_distortion*: maximum attack manipulation; *stepsize* : size of modification in each iteration

$\mathbf{x}' \leftarrow \mathbf{x}$
$iter \leftarrow 0$
while $y = f(\mathbf{x}')$ and $iter \leq max\_iter$ **do**
  $\mathbf{u} \leftarrow \frac{\nabla f_y(\mathbf{x}')}{||\nabla f_y(\mathbf{x}')||_2}$
  $\mathbf{x}^t \leftarrow \mathbf{x}' - stepsize \times \mathbf{u}$
  if $d(\mathbf{x}, \mathbf{x}^t) \geq max\_distortion$ **then**
    break
  end if
  $\mathbf{x}' \leftarrow \mathbf{x}^t$
end while
return $\mathbf{x}'$

Fig. 4 shows an example of an original and its attacked sample generated by Algorithm 1. A conventional stacked autoencoder containing two hidden layers with 600–400 neurons is trained using MNIST dataset [41]. The left image shows a handwriting digit



(a) Original Image (b) Attacked Image

Fig. 4. Example of original and its attacked image generated by evasion attack.

five, which can be correctly classified by the stacked autoencoder. The original image is then manipulated according to Algorithm 1, and the attacked image is shown on the right hand side. Although the digital five is still clearly shown in the attacked image, it is recognized as the digit three by the stacked autoencoder.

5. Experiment

In this section, the performance of our proposed methods is evaluated and compared with existing autoencoders in terms of error rate, cost of attack, output stability, and time complexity under the evasion attack. Then, a preliminary investigation on the robustness of our methods using contaminated dataset is also discussed.

5.1. Experiment settings

Two datasets, MNIST and SVHN dataset [41], are used in our experiments. MNIST contains 70,000 grayscale images from 10 classes, while the 99,289 color images from 10 classes in SVHN dataset are firstly converted into grayscale images. All feature values are firstly normalized to [0, 1] by $v_n = (v - min_v)/(max_v - min_v)$, where v and v_n denote the original and the normalized values, while max_v and min_v are the maximum and minimum values of the feature. For each experiment, a dataset is firstly divided into two sets: 80% for training set and 20% for test set. 20% of training set is reserved as the validation set to control the stop of training.

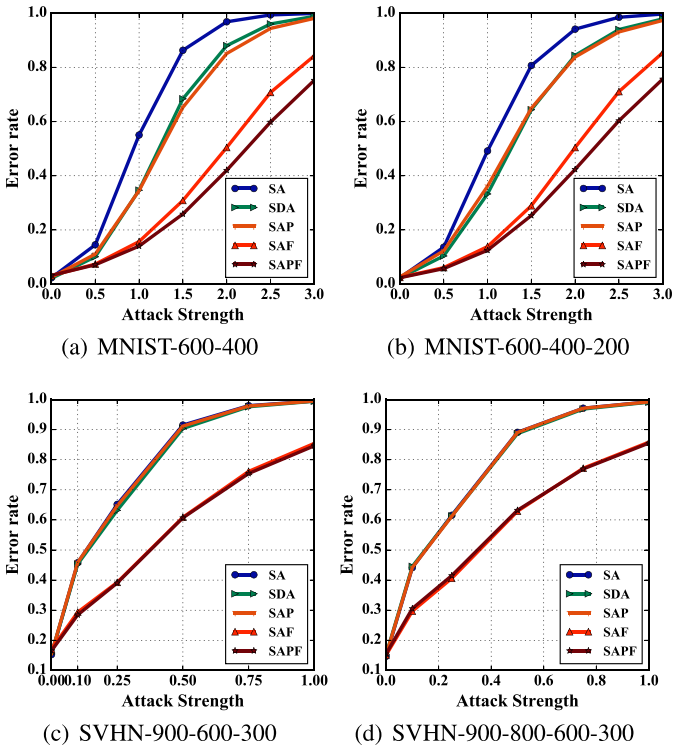
The experiments are performed on a computer with 32 GB of memory and two processors: Intel processor with E5-1620U v2 cores at 3.70 GHz and GPU Quadro K2000. For our methods, the sensitivity measure is considered in the unsupervised representation learning phase (*SAP*), the supervised fine-tuning phase (*SAF*) and both phases (*SAPF*) of the stacked autoencoder. Existing methods including the stacked autoencoder(*SA*) and the stacked denosing autoencoder(*SDA*) are used as benchmarks to compare with our methods. Architectures of all methods are the same and the only difference is the learning algorithm. Softmax regression is used in the output layer for classification. Maximum epochs of iterations is set as 500 and 2,000 for MNIST and SVHN dataset, respectively. Two architectures are used in MINST: one contains two hidden layers with 600 and 400 neurons (MINST-600-400), and another one contains three hidden layers with 600, 400 and 300 neurons (MINST-600-400-200). On the other hand, two more complicated architectures are chosen for SVHN due to its complexity [41]: three hidden layers with 900, 600 and 300 neurons (SVHN-900-600-300), and four hidden layers with 900, 800, 600, and 300 neurons (SVHN-900-800-600-300).

The parameters of different methods are determined according to a preliminary evaluation aiming to maximize their average accuracies. For the proposed methods, pre-selected values of the trade-off parameter λ, including 0.1, 0.5, 1 and 5, and the size of the Q-neighborhood q, including 0.05, 0.1, 0.3 and 0.5, have been evaluated in the preliminary evaluation. For the evasion attack, denoising level is set to 0.5 and 0.1 for MNIST and SVHN dataset for *SDA*, respectively. λ and q are 1 and 0.3 for MNIST, and 1 and 0.1 for SVHN dataset in our methods, respectively. To avoid long training time, three samples are used in the Monte Carlo method. On the other hand, denoising level is set to 0.1 in *SDA* in the experiment on the contaminated training set. Our methods also set $\lambda = 1$, $q = 0.1$ and three artificial samples in the Monte Carlo method. All classifiers are implemented in the python environment by Theano [42]. Each experiment is executed five times independently.

The evasion attack is simulated according to Algorithm 1 in Section 4. The cost of the evasion attack is represented by attack strength defined as the Euclidean distance between an original sample ($\mathbf{x}$) and an adversarial sample ($\mathbf{x}'$) modified from $\mathbf{x}$. The attack strength is set as 0, 0.5, 1.0, 1.5, 2.0, 2.5, and 3.0 in MNIST, and 0.0, 0.25, 0.5, 0.75, and 1 in SVHN separately. On the other hand, Section 5.3 will also evaluate the robustness of the autoencoders in learning a contaminated dataset. Gaussian noise with zero mean and standard deviations 0.1 and 0.2 are added to all samples used in training. We assume that all methods are in the worst scenario in which an adversary has complete knowledge on the targeted classifier, *i.e.*, the decision function and all its parameters are known.

Table 1
Error rate (%) of models with different training algorithms without attack.

Settings	SA	SDA	SAP	SAF	SAPF
MNIST-600-400	2.34	2.13	2.52	3.15	3.24
MNIST-600-400-200	2.35	2.27	2.65	2.46	2.36
SVHN-900-600-300	15.26	15.50	15.52	17.07	16.54
SVHN-900-800-600-300	14.68	15.20	14.98	15.08	15.50



(a) MNIST-600-400 (b) MNIST-600-400-200

(c) SVHN-900-600-300 (d) SVHN-900-800-600-300

Fig. 5. Error rate of models under evasion attack.

5.2. Experimental result on evasion attack

5.2.1. Error rate

When there is no attack, the accuracies of all methods are between 2% and 4% on MNIST and 15–17% on SVHN as shown in Table 1. Our proposed methods, *i.e., SAP, SAF* and *SAPF*, insignificantly perform worse than *SA* and *SDA* in MNIST. With a more complicated architecture, the errors of *SAF* and *SAPF* drop slightly. Similar results can be observed in SVHN, when a simpler architecture is applied, the error rates of our proposed methods are higher than the traditional ones. However, all performances are similar when the architecture is more complicated.

The average error of the stacked autoencoder methods with architectures under different attack strengths in MINST and SVHN are shown in Fig. 5. *SA* has the worst performance among all methods under the evasion attack shown in Fig. 5(a). When the attack strength increases, the error of *SA* rises dramatically in both architectures. When the attack strength is 1.0, *SA* makes a wrong decision with a 50% chance. However, *SDA* is more robust than

Table 2
Average cost for successful evasion attack over all attack samples.

Setting	SA	SDA	SAP	SAF	SAPF
MNIST-600-400	1.05	1.34	1.38	2.11	2.37
MNIST-600-400-200	1.13	1.40	1.39	2.10	2.35
SVHN-900-600-300	0.28	0.29	0.28	0.60	0.61
SVHN-900-800-600-300	0.30	0.31	0.30	0.58	0.57

SA. It indicates that considering noise distortion in the representation learning increases the robustness. This result is consistent with those obtained in [35]. The performance of *SDA* is similar to our *SPA* since both of them revise the algorithm of the representation learning. In the architecture 600-400-200, their errors are nearly the same in all attack situations. On the other hand, *SAF*, which adds the sensitivity measure in fine-tuning learning phase, performs much more robustly than *SPA*. For example, when the attack strength is 1.0, the error rate of *SAF* is less than 20% while *SAP* is more than 30% in both architectures. This suggests that considering sensitivity measure in the fine-tuning phase is more effective than the one in the representation learning phase. When sensitivity measure is applied in both phases, *SAPF*, has the lowest error rate among all methods in all attack situations. The error rate increases gently with the increase in the attack strength. Especially when attack strength is 1.5 and 2.0, the error rate of *SAPF* is only half of the ones of *SA, SDA* and *SPA*. Fig. 5(b) indicates that all methods with three hidden layers behave consistently with the ones with two hidden layers in all situations.

The performance of the methods in SVHN are shown in Fig. 5(c) and (d). All methods are more vulnerable in SVHN than the ones in MNIST, *i.e.,* the error rate of a method is higher in SVHN than the one in MNIST under evasion attack with the same attack strength. For example, when the attack strength is 1.0, *SA* is nearly completely wrong in SVHN but in MNIST, its accuracy is more than 40%. *SA, SDA* and *SPA* perform similarly under different attack strength, and these methods suffer from evasion attack significantly. In contrast, *SAF* and *SAPF* achieves similar performance, which is more robust than *SA, SDA* and *SPA*. Considering the sensitivity in representation learning cannot improve the robustness of autoencoder for this dataset.

5.2.2. Robustness

Hardness of evasion [43,44] is defined as the average cost for successful evasion attack over all attack samples. Euclidean distance is applied to measure the attack cost. Longer distance means that more modification is required for a successful evasion, which means a classifier is harder to be evaded. Table 2 shows *hardness of evasion* of the methods with different architectures in different datasets.

In Table 2, the cost of successful evasion attack to *SAPF* is the largest among all methods in all settings, which means *SAPF* is the most robust. There is a large difference between the cost of the methods which consider and those that do not consider sensitivity in the fine-tuning phase. For example, the cost of *SA, SDA* and *SAP* is 1.05–1.40 in MNIST dataset and 0.28–0.31 in SVHN dataset, while *SAF* and *SAPF* require 2.10–2.35 in MNIST and 0.58–0.61 for successful evasion attack in SVHN. It confirms that fine-tuning phase with sensitivity increases the robustness of the autoencoder. In addition, the average costs of successful attack of *SAPF* are more than the ones of *SAF* except in SVHN-900-800-600-300 in which the costs are equal. This indicates that considering the sensitivity in both the representation learning and the fine-tuning phases is more secure than the ones only considering the sensitivity in fine-tuning.

The influence of an evasion attack on the outputs of different kinds of autoencoders is also evaluated. We fix the attack cost defined as the number of iteration in Algorithm 1 as 10 for MNIST and 5 for SVHN. The average decline of the confidence on the real class of samples, *i.e.,* the output of a classifier on the real class, is measured. The average decline is shown in Table 3.

Table 3
Average confidence decline (%) after a fixed number of iteration in evasion attack.

Setting	SA	SDA	SAP	SAF	SAPF
MNIST-600-400	26.20	17.78	21.53	10.21	11.39
MNIST-600-400-200	23.46	17.13	22.36	9.42	10.45
SVHN-900-600-300	69.43	68.76	69.05	29.66	29.19
SVHN-900-800-600-300	65.73	65.08	52.37	33.70	33.75

Table 4
Average output fluctuation of the models with 600–400 in MNIST.

MNIST-600-400	SA	SDA	SAP	SAF	SAPF
Layer #1	9.11	8.29	3.54	6.94	2.77
Layer #2	5.23	4.64	2.42	2.85	1.06
Output layer	0.63	0.53	0.41	0.12	0.08

Table 5
Average output fluctuation of the models with 600–400–200 in MNIST.

MNIST-600-400-200	SA	SDA	SAP	SAF	SAPF
Layer #1	9.13	8.08	3.42	7.66	3.37
Layer #2	5.20	4.35	2.02	3.48	1.76
Layer #3	3.88	3.88	1.32	1.68	0.50
Output layer	0.59	0.47	0.37	0.07	0.07

Table 6
Average output fluctuation of the models with 900-600-300 in SVHN.

SVHN-900-600-300	SA	SDA	SAP	SAF	SAPF
Layer #1	8.26	7.04	7.00	7.82	6.31
Layer #2	7.60	6.66	6.65	7.05	5.62
Layer #3	6.50	6.11	6.12	4.41	3.99
Output layer	0.98	0.90	0.87	0.31	0.31

Table 7
Average output fluctuation of the models with 900-800-600-300 in SVHN.

SVHN-900-800-600-300	SA	SDA	SAP	SAF	SAPF
Layer #1	8.18	6.93	7.50	8.27	7.74
Layer #2	7.74	6.51	7.23	8.52	7.50
Layer #3	7.84	7.05	7.62	7.02	6.38
Layer #4	5.53	5.38	5.62	2.94	3.38
Output layer	0.92	0.81	0.94	0.54	0.49

The average confidence decline of *SAF* and *SAPF* is less than half of *SA* and *SDA* in most cases. For example, *SAF* and *SAPF* are less than 28% while other methods are higher than 70% in SVHN-900-600-300. However, the confidence decline on *SAP* is not significantly lower than *SA*, and sometimes even larger than *SDA*. These results are consistent with the previous ones, which only considering the sensitivity in the presentation learning phase may not increase the robustness significantly.

Moreover, we investigate how a small change ($\Delta\mathbf{x}$) in input features will affect the output of a hidden layer and the output layer of the methods, *i.e., dist*$(f(\mathbf{x}), f(\mathbf{x}+\Delta\mathbf{x}))$, where *dist* is a function measuring the distance, and f is the function of a hidden layer or the output layer. We use Euclidean distance as *dist* in this experiment, and $\Delta\mathbf{x}$ is a random variable following the Gaussian distribution with 0 mean and 0.5 standard deviation.

Average output fluctuation values of different layers of the several methods are given in Tables 4–7. Unsurprisingly, as *SA* only minimizes the error, its output fluctuation of all layers is the highest among all models in most cases. It explains why our previous experimental results show that *SA* is vulnerable since the output

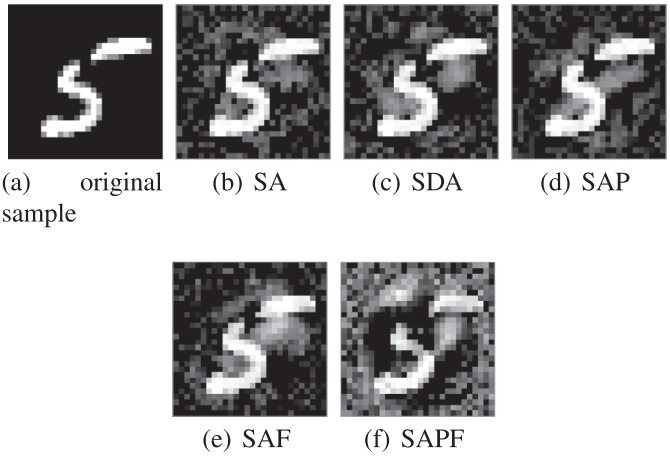


Fig. 6. A successful attacked sample with minimum modification.

of *SA* may change significantly with a small change in the input. The fluctuation of *SA* is reduced by considering noise in training in *SDA*. On the other hand, as the sensitivity is minimized in the representation learning in all hidden layers of *SAP*, the output fluctuation of hidden layers of *SAP* is relatively smaller than the ones of *SAF*. *SAPF*, which considers the sensitivity in both representation learning and fine-tuning phases, achieves the lowest fluctuation in all layers generally. It explains why *SAPF* achieves excellent results against evasion attack.

A successfully evaded sample with minimum modification for each method with 600–400 architecture on MNIST dataset is shown in Fig. 6. The images with 28*28 pixels are in the grey scale. Fig. 6(a) shows the original sample which is a digit five. Evasion attack changes a pixel closer to white or black color by modifying its value in order to mislead a classifier. Obviously, the successfully evaded sample to *SAPF* is more difficult to recognize as a digit five than the evaded samples of other methods, *i.e.,* more manipulation is required for successfully evasion to *SAPF*.

5.2.3. Time complexity

The time complexity of *SA* and *SDA* is in the same big-O, which is in $O(\sum_{l=1}^{d} n_{k-1} \times n_k \times m \times I_{max,l}$, where d denotes the number of layers in a network, n_k is the number of neurons in the kth layer, m is the number of samples ,and $I_{max,\,l}$ denotes the maximum iteration of gradient descent in the kth layer. However, the calculation of the sensitivity measure increases the complexity of our proposed methods, such as *SAP, SAF* and *SAPF*. The time complexity of our methods is in $O(\sum_{l=1}^{d} n_{k-1} \times n_k \times m \times I_{max,l} \times s)$, where s is the number of artificial samples used in the Monte Carlo method for sensitivity measure approximation. Although using more artificial samples improves the accuracy of approximation in general, the learning process takes longer time. For example, three samples are used in this experiment. The running times of our methods are roughly three times than the ones of *SA* and *SDA*. As the training process is often carried out offine, the additional running time required by our methods may not be critical.

5.3. Experimental result on contaminated training set

The performance of our proposed learning algorithm on contaminated training set is also evaluated. This preliminary study investigates the robustness of our methods (*SAP, SAF* and *SAPF*) and the existing autoencoders (*SA* and *SDA*) under random Gaussian noise attack in MNIST. The experimental results of different attack strengths, with the standard deviation of Gaussian noise equal to 0.0, 0.1 and 0.2, are shown in Fig. 7. The range of y-axis is from

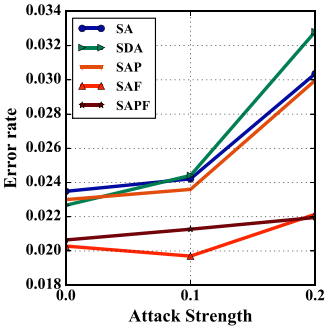


Fig. 7. Error rate of the models trained by the dataset contaminated by Gaussian noise with different standard deviation.

1.8% to 3.4% in order to have better illustration. The accuracies of all methods drop slightly with increase of the attack strength, which indicates that the stacked autoencoder does not significantly suffer from a training set contaminated with the Gaussian noise. However, the error rates of our methods, *i.e., SAP, SAF* and *SAPF* increase more gently than the ones of *SA* and *SDA*, especially for *SAF* and *SAPF*. When the standard deviation of the Gaussian noise increases from 0.0 to 0.2, the error rates of *SAF* and *SAPF* only increase around 0.2%, respectively. The result suggests that *SAPF* is the most robust among all these methods when the training set is contaminated by the Gaussian noise.

5.4. Discussion

The criteria including the error rate, the cost of successful evasion, the decline of output confidence, and the fluctuation of output, which are used in the experiments, indicate consistently that our proposed methods are more robust than *SA* and *SDA* under the evasion attack. The results suggest that learning with the sensitivity term improves the robustness of the stacked autoencoder against the evasion attack. However, as the results of *SAP* are only slightly better than *SDA* and sometimes worse than *SDA*, only considering the sensitivity in the representation learning phase does not contribute to the robustness effectively. On the other hand, for the stacked autoencoder with the sensitivity in the fine-tuning phase, *SAF*, has more robust performance in comparison with the traditional methods. When training with the sensitivity in both representation learning and fine-tuning, *SAPF* is the most robust among all methods in all scenarios. It confirms that the sensitivity is a critical factor to reduce the influence of an evasion attack to the stacked autoencoder in an adversarial environment.

The tradeoff of robustness is the time complexity. The running time of our methods is larger than *SA* and *SDA* due to the calculation of the additional sensitivity term in the training process. The Monte Carlo method applied in this paper estimates the value of the sensitivity by using artificial samples, which increases the running time of our methods. Since the models can be trained offline, the time complexity is not a critical problem.

A preliminary study on how a contaminated dataset affects our models is also carried out. Our methods, especially *SAF* and *SAFP*, are less influenced by the Gaussian noise than the traditional methods, *i.e.,* the error rates of the proposed methods increase more slowly than the others with the increase of standard deviation of the Gaussian noise.

6. Conclusion

In this paper, we proposed a robust learning algorithm which makes use of the sensitivity measure for the stacked autoencoder against an evasion attack. To the best of our knowledge, there is

no study on the robust learning for the stacked autoencoder in an adversarial environment. Our method trains the autoencoder by minimizing not only the error but also the sensitivity defined as the change of the output due to a small fluctuation on the input. Our methods can be applied to the unsupervised representation learning and the supervised fine-tuning. Besides accuracy, the performances of our methods are evaluated experimentally based on the hardness of evasion, the decline of output confidence and the output fluctuation caused by an evasion attack. All criteria suggest that considering the sensitivity in both the representation learning and the fine-tuning phases are the most robust in comparison with the conventional stacked autoencoder, denoising stacked autoencoder, and our methods which only apply the sensitivity in either the representation learning or the fine-tuning phase, but not both. The results show that the importance of considering sensitivity in robust learning for the stacked autocencoder against evasion attack.

The experiment also indicates that our proposed methods are more stable on noisy training samples than the traditional methods. As a result, influence of attack samples in a training set to a classifier might be reduced by considering sensitivity measure in training. A further investigation on the performance of the proposed methods under different kinds of poisoning attacks should be carried out.

One drawback of our methods is the time complexity. Longer training time is required by our methods than the traditional ones since the Monte-Carlo method is time consuming. One possible solution of this limitation is to calculate the sensitivity term by using mathematical approximation with a lower time complexity. However, the performance of our methods may be scarified due to the inaccurate calculation of sensitivity. Another interesting research direction is how to determine the size of the Q-neighborhood. One characteristic of the Q-neighborhood is its geometric meaning. A larger size of the Q-neighborhood covers a larger region but has lots of overlapping. A proper size value should be application dependent. The determination of this value will be helpful in a better understanding of the application problem. Data complexity which measures the difficulty of classification may be an useful tool to identify the size of the Q-neighborhood.

Sensitivity measure is a general concept and it has been applied to many different applications successfully, e.g. feature selection and classifier fusion. Our model might also improve the security of other deep learning networks in an adversarial environment. However, how to measure the change of the output of other deep neural networks caused by the fluctuation of input should be further investigated. Moreover, whether and how robustness in learning affects the generalization ability of other models is another interesting problem that desverves a further investigation.

To conclude, we believe that this study is the first investigation on the robustness of the stacked autoencoder under an evasion attack. This study shows that the sensitivity term is helpful in improving the security of the stacked autoencoder.

Acknowledgments

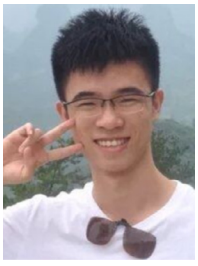
This work is supported by the Fundamental Research Funds for the Central Universities (2015ZZ092).

References

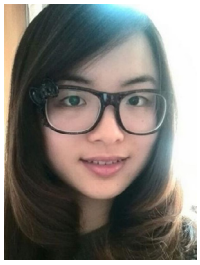
[1] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2016) 11–26.
[2] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the International Conference on Neural Information Processing Systems, 2012, pp. 1097–1105.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the Computer Vision and Pattern Recognition, 2014, pp. 1–9.
[4] J.J. Tompson, A. Jain, Y. LeCun, C. Bregler, Joint training of a convolutional network and a graphical model for human pose estimation, in: Advances in neural information processing systems, 2014, pp. 1799–1807.
[5] N. Zeng, Z. Wang, H. Zhang, W. Liu, F.E. Alsaadi, Deep belief networks for quantitative analysis of a gold immunochromatographic strip, Cogn. Comput. 8 (4) (2016) 684–692.
[6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (1) (2011) 2493–2537.
[7] S. Jean, K. Cho, R. Memisevic, Y. Bengio, On using very large target vocabulary for neural machine translation, arXiv preprint arXiv:1412.2007. (2013).
[8] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97.
[9] T.N. Sainath, A.R. Mohamed, B. Kingsbury, B. Ramabhadran, Deep convolutional neural networks for lvcsr, in: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 8614–8618.
[10] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Adv. Neural Inf. Process. Syst. 19 (2007) 153.
[11] D. Lowd, C. Meek, Good word attacks on statistical spam filters, in: In Proceedings of the Second Conference on Email and Anti-Spam (CEAS), 2005.
[12] B. Biggio, Z. Akhtar, G. Fumera, G.L. Marcialis, F. Roli, Security evaluation of biometric authentication systems under real spoofing attacks, IET Biom. 1 (1) (2012) 11–24.
[13] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, in: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 387–402.
[14] F. Zhang, P.P. Chan, B. Biggio, D.S. Yeung, F. Roli, Adversarial feature selection against evasion attacks, IEEE Trans. Cybern. 46 (3) (2016) 766–777.
[15] A. Demontis, P. Russu, B. Biggio, G. Fumera, F. Roli, On security and sparsity of linear classifiers for adversarial settings, in: Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2016, pp. 322–332.
[16] H. Xiao, B. Biggio, B. Nelson, H. Xiao, C. Eckert, F. Roli, Support vector machines under adversarial label contamination, Neurocomputing 160 (2015) 53–62.
[17] P.P. Chan, C. Yang, D.S. Yeung, W.W. Ng, Spam filtering for short messages in adversarial environment, Neurocomputing 155 (2015) 167–176.
[18] X. Jia, X. Yang, K. Cao, Y. Zang, N. Zhang, R. Dai, X. Zhu, J. Tian, Multi-scale local binary pattern with filters for spoof fingerprint detection, Inf. Sci. 268 (2014) 91–102.
[19] P. Louvieris, N. Clewley, X. Liu, Effects-based feature identification for network intrusion detection, Neurocomputing 121 (2013) 265–273.
[20] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv preprint arXiv: 1312.6199. (2013).
[21] A. Graese, A. Rozsa, T.E. Boult, Assessing threat of adversarial examples on deep neural networks, arXiv preprint arXiv: 1610.04256. (2016).
[22] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv: 1412.6572. (2014).
[23] Q. Wang, W. Guo, K. Zhang, X. Xing, C.L. Giles, X. Liu, Random feature nullification for adversary resistant deep architecture, arXiv preprint arXiv: 1610.01239. (2016).
[24] Q. Wang, W. Guo, I. Ororbia, G. Alexander, X. Xing, L. Lin, C.L. Giles, X. Liu, P. Liu, G. Xiong, Using non-invertible data transformations to build adversary-resistant deep neural networks, arXiv preprint arXiv: 1610.01934. (2016).
[25] W.W. Ng, Z.-M. He, D.S. Yeung, P.P. Chan, Steganalysis classifier training via minimizing sensitivity for different imaging sources, Inf. Sci. 281 (2014) 211–224.
[26] P.P. Chan, D.S. Yeung, W.W. Ng, C.M. Lin, J.N. Liu, Dynamic fusion method using localized generalization error model, Inf. Sci. 217 (2012) 1–20.
[27] D.S. Yeung, W.W. Ng, D. Wang, E.C. Tsang, X.-Z. Wang, Localized generalization error model and its application to architecture selection for radial basis function neural network, IEEE Trans. Neural Netw. 18 (5) (2007) 1294–1305.
[28] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: IEEE Symposium on Security and Privacy (SP) 2017, 2017, pp. 39–57.
[29] M. Barreno, B. Nelson, A.D. Joseph, J. Tygar, The security of machine learning, Mach. Learn. 81 (2) (2010) 121–148.
[30] M. Barreno, B. Nelson, R. Sears, A.D. Joseph, J.D. Tygar, Can machine learning be secure? in: Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security, ACM, 2006, pp. 16–25.
[31] L. Huang, A.D. Joseph, B. Nelson, B.I. Rubinstein, J. Tygar, Adversarial machine learning, in: Proceedings of the 4th ACM workshop on Security and artificial intelligence, ACM, 2011, pp. 43–58.
[32] P.P.K. Chan, Z.-M. He, H. Li, C.-C. Hsu, Data sanitization against adversarial label contamination based on data complexity, Int. J. Mach. Learn. Cybern. (2017) 1–14, doi:10.1007/s13042-016-0629-5.
[33] Y. Bengio, Learning deep architectures for AI, Found. Trends® Mach. Learn. 2 (1) (2009) 1–127.
[34] H. Larochelle, Y. Bengio, J. Louradour, P. Lamblin, Exploring strategies for training deep neural networks., J. Mach. Learn. Res. 10 (10) (2009) 1–40.
[35] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th international conference on Machine learning, ACM, 2008, pp. 1096–1103.

[36] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[37] Y. Qi, Y. Wang, X. Zheng, Z. Wu, Robust feature learning by stacked autoencoder with maximum correntropy criterion, in: Proceedings of the ICASSP 2014 - 2014 IEEE International Conference on Acoustics, Speech and Signal Processing, 2014, pp. 6716–6720.
[38] D.S. Yeung, P.P. Chan, W.W. Ng, Radial basis function network learning using localized generalization error bound, Inf. Sci. 179 (19) (2009) 3199–3217.
[39] N. Metropolis, S. Ulam, The Monte carlo method, J. Am. Stat. Assoc. 44 (247) (1949) 335–341.
[40] P. Russu, A. Demontis, B. Biggio, G. Fumera, F. Roli, Secure kernel machines against evasion attacks, in: Proceedings of the ACM Workshop on Artificial Intelligence and Security, 2016, pp. 59–69.
[41] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A.Y. Ng, Reading digits in natural images with unsupervised feature learning, in: Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning, Vol. 2011, 2011, p. 5.
[42] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al., Theano: a Python framework for fast computation of mathematical expressions, arXiv e-prints arXiv: abs/1605.02688. (2016).
[43] B. Biggio, G. Fumera, F. Roli, Multiple classifier systems for adversarial classification tasks, in: Proceedings of the International Workshop on Multiple Classifier Systems, Springer, 2009, pp. 132–141.
[44] B. Biggio, G. Fumera, F. Roli, Adversarial pattern classification using multiple classifiers and randomisation, in: Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2008, pp. 500–509.

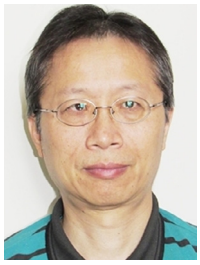
Patrick P.K. Chan received the Ph.D. degree from Hong Kong Polytechnic University in 2009. He is currently Associate Professor of School of Computer Science and Engineering, and the person in charge of Machine Learning and Cybernetics Research Laboratory in South China University of Technology, Guangzhou, China. He is also a part-time Lecturer of Hyogo College of Medicine, Japan. His current research interests include pattern recognition, multiple classifier system, biometric, computer security, deep learning and reinforcement learning. Dr. Chan was a member of the governing boards of IEEE SMC Society 14–16 and also the Chairman of IEEE SMCS Hong Kong Chapters 14–15. He is the counselor of IEEE Student Branch in South China University of Technology. He serves as an organizing committee chair of several international conferences, and an associate editor for international journals including Information Sciences, and International Journal of Machine Learning and Cybernetics.



Zhe Lin received B.Sc. degree in Computer Science from South China University of Technology, Guangzhou, China in 2016. He is currently a master student of School of Computer Science and Engineering, and a member of Machine Learning and Cybernetics Research Laboratory in South China University of Technology, Guangzhou, China. His current research interests include deep learning, robust learning, images recognition and objects detection.



Xian Hu received B.Sc. degree in Computer Science from Beijing Language and Culture University, Beijing, China in June, 2012. From July, 2012 to May, 2015, she was an algorithm engineer in Vion-Tech, Beijing, China. She is currently a master student of School of Computer Science and Engineering, and a member of Machine Learning and Cybernetics Research Laboratory in South China University of Technology, Guangzhou, China. Sirly has rich experience in computer vision such as object recognition, tracking, and behaviors analysis. Her current research interests include deep learning, reinforcement learning, computer vision, and computer security.



Eric C. C. Tsang received the B.S. degree in computer studies from the City University of Hong Kong, Hong Kong, in 1990, and the Ph.D. degree in computing from Hong Kong Polytechnic University, Hong Kong, in 1996. He is an Associate Professor with the Faculty of Information Technology, Macau University of Science and Technology, Macau, China. His current research interests include fuzzy systems, rough sets, fuzzy rough sets, multiple classifier systems and deep learning.

Daniel S. Yeung received his Ph.D. in applied mathematics from Case Western Reserve University, USA. He was an assistant professor at the Department of Mathematics and later the School of Computer Science and Technology at Rochester Institute of Technology, Rochester, New York from 1974–1980. In the next ten years He held industrial and business positions as a Technical Specialist/Application Software Group Leader at the Computer Consoles, Inc., Rochester, New York, an Information Resource Sub-manager/Staff Engineer at the Military and Avionics Division, TRW Inc., San Diego, California, and an Information Scientist of the Information System Operation Lab, General Electric Corporate Research and Development Centre, Schenectady, New York. In 1989 he joined City Polytechnic of Hong Kong as an Associate Head/Principal Lecturer at the Department of Computer Science. Then he served as the founding Head and Chair Professor of the Department of Computing at The Hong Kong Polytechnic University until his retirement at 2006. Currently he is a Chair Professor in the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China. Dr. Yeung is a fellow of the IEEE and served as the President of IEEE SMC Society in 2008–09. His current research interests include neural-network sensitivity analysis, large scale data retrieval problem and cyber security.