An Autoencoder-based Method for Targeted Attack on Deep Neural Network Models

Duc-Anh Nguyen
VNU University of Engineering and Technology
nguyenducanh@vnu.edu.vn

Do Minh Kha
VNU University of Engineering and Technology
17020827@vnu.edu.vn

Pham Thi To Nga
Dai Nam University
ngaptt@dainam.edu.vn

Pham Ngoc Hung
VNU University of Engineering and Technology
hungpn@vnu.edu.vn

***Abstract*—This paper presents an autoencoder-based method for a targeted attack on deep neural network models, named AE4DNN. The proposed method aims to improve the existing targeted attacks in terms of their generalization, transferability, and the trade-off between the quality of adversarial examples and the computational cost. The idea of AE4DNN is that an autoencoder model is trained from a balanced subset of the training set. The trained autoencoder model is then used to generate adversarial examples from the remaining subset of the training set, produce adversarial examples from new samples, and attack other DNN models. To demonstrate the effectiveness of AE4DNN, the compared methods are box-constrained L-BFGS, Carlini-Wagner $||L||_2$ attack, and AAE. The comprehensive experiment on MNIST has shown that AE4DNN can gain a better transferability, improve generalization, and generate high quality of adversarial examples while requiring a low cost of computation. This initial result demonstrates the potential ability of AE4DNN in practice, which would help to reduce the effort of testing deep neural network models.**

***Index Terms*—robustness, adversarial example generation, targeted attack, white-box testing, transferability, generalization.**

I. Introduction

Deep learning [1], known as a subset of machine learning, has been used widely in solving many problems related to classification, ranging from speech processing [2], image processing [3], to medical diagnosis [4], etc. A deep neural network (DNN) model is commonly built using Pytorch, Tensorflow, Scikit-learn, Keras, and Caffe. In most of the cases, if DNN models are trained carefully with appropriate hyperparameters, the accuracy of these models usually outperforms the traditional machine learning methods such as Naive Bayes [5], K-nearest neighbours (KNN), etc. However, it is not simple to explain the properties of DNN models such as the value of weights, or the role of neurons in different layers, etc. Due to the unexplainable characteristic of DNN models, the testing of DNN models should be taken into account carefully to ensure the robustness of these models.

Targeted adversarial example generation (or targeted adversarial attack) in the white-box approach is a new trend to test the robustness of DNN models by generating adversarial examples for these models [6]. An adversarial example is a slight modification of an input vector in the training set to make the model label this adversarial example as a specific class [7]. A targeted adversarial generator needs to know the training set, the architecture, and the weights of the attacked model [6]. The general problem of targeted adversarial example generation is stated as follows. Given a training set consisting of input vectors, an attacked DNN model trained on this training set, and a target label, every original input vector is modified slightly to generate adversarial examples which would be classified as the target label. There are many popular metrics such as $||L||_2$ norm and neuron coverage [8], etc. used to evaluate the robustness of a targeted adversarial example generation method.

However, in terms of $||L||_2$, the targeted adversarial example generation methods in the white-box approach usually have three main problems. The first problem is the transferable rate. The generated adversarial examples from a DNN model should be reused to check other DNN models to reduce the cost of DNN testing. The second issue is related to generalization. Specifically, the result of previous attacks on a DNN model should be reused for future attacks on that model, which contributes to the decrease in the cost of targeted attack. Although adversarial autoencoding ATN [9] provides this capability, this method still seems to work less effectively. The last problem is the trade-off between the quality of adversarial examples and the computational cost. Some other $||L||_2$ attack methods can generate high quality of adversarial examples but they usually consume high computational cost such as Carlini-Wagner $||L||_2$ attack [10] and box-constrained L-BFGS [11].

Therefore, this paper proposes a targeted adversarial example generation method in the white-box approach, named AE4DNN, to mitigate the problems mentioned above. The objective study is the popular training set MNIST [12]. The generated adversarial examples should be close to its input vector under $||L||_2$ distance metric. The key idea of AE4DNN is that given a balanced subset of the training set and a target label, an autoencoder will be trained to find out the best matrix of weights. Here, the objective function includes two

978-1-6654-0435-8/21/$31.00 ©2021 IEEE

parts, in which each has its weight. The first one minimizes the sum of squared $||L||_2$ distances between input vectors and their corresponding reconstructed vectors. The second part takes responsibility for minimizing the sum of cross-entropy between the prediction of reconstructed vectors and their corresponding target vectors. We propose a formula to specify the good range of the weight between these two parts, which might reduce the effort of configuration selection. The trained autoencoder is then used to generate adversarial examples from the remaining subset of the training set or new input vectors (i.e. generalization ability) with high quality of adversarial example while requiring low computational cost (i.e. trade-off mitigation). Additionally, this autoencoder could be used to attack other DNN models (i.e. transferability).

The rest of this paper is organized as follows. Section II provides the background of DNN and well-known targeted adversarial example generation methods in the white-box approach. The overview of AE4DNN is shown in Sect. III. Next, Section IV presents the experiment to prove the effectiveness of AE4DNN in terms of generalization, transferability, and trade-off mitigation. Section V then gives the overview of related research about adversarial example generation for DNN model. Finally, the conclusion and future work are described in Sect. VI.

II. Background

A. Deep Neural Network

Deep neural network (DNN). A deep neural network is a 3-tuple $M = (\mathbf{L}, \mathbf{W}, \boldsymbol{\theta})$, where $L = [l_0, l_1, ..., l_{h-1}]$ is a matrix of layers and h is the number of layers. $\mathbf{W} = [\mathbf{w}_{0,1}, \mathbf{w}_{1,2}, .., \mathbf{w}_{l-2,l-1}]$ is a matrix of weights, in which $\mathbf{w}_{i,i+1}$ represents the weights between the i^{th} layer and the $(i+1)^{th}$ layer. $\boldsymbol{\theta} = [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_{h-1}]$ is a matrix of activation functions in which each activation belongs to a layer.

For simplicity, a set of neurons on the layer l_i $(i \in [0, .., h-1])$ is denoted with n_i. According to this convention, a set of neurons on the input layer and on the output layer of a DNN are denoted with n_0 and n_{h-1}, respectively. The paper denotes the number of input vectors by s, the number of input features by d, the number of classes by k, and hyper-parameters by ζ. Based on the characteristics of the output layer of DNN, the number of neurons on the output layer n_{h-1} is equal to k.

Denote an input vector by $\mathbf{x} \in \mathbf{X}^{s \times d}$, where $\mathbf{X}$ is the training set. For a specific neuron, let $n_i^j(\mathbf{x}) \in \mathbb{R}$ denote the value of the j^{th} neuron on layer l_i of the attacked DNN with input vector $\mathbf{x}$. The true probability vector of $\mathbf{x}$ is denoted by $\mathbf{y}_{true} \in \mathbb{R}^k$. The prediction vector of $\mathbf{x}$ is one dimensional array, denoted by $\mathbf{y} = [n_{h-1}^0, n_{h-1}^1, .., n_{h-1}^{k-1}]^T \in \mathbb{R}^k$. The predicted label of $\mathbf{x}$ is denoted by $c_{\mathbf{x}}$, where $c_{\mathbf{x}} = \text{argmax}_{i \in [0..k-1]}\{n_{h-1}^i\}$.

Given $\mathbf{x}$, let $f(\mathbf{x}, \mathbf{y}_{true}, \zeta) \in \mathbb{R}$ denote the objective function of a DNN used for the training purpose. $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y}_{true}, \zeta) \in \mathbb{R}^d$ is the gradient of the objective function with respect to $\mathbf{x}$.

B. Adversarial Example

In the white-box testing approach, adversarial example in targeted attack is defined as follows. Given a DNN $M = (\mathbf{L}, \mathbf{W}, \boldsymbol{\theta})$, an input vector $\mathbf{x}$, and a target label y^*, a modified vector $\mathbf{x}'$ from $\mathbf{x}$ is an adversarial example iff it satisfies all the following requirements:
(i) $\mathbf{x}$ is classified correctly,
(ii) y^* is different from $c_{\mathbf{x}}$,
(iii) the classification of $\mathbf{x}'$ is equal to y^*, and
(iv) the distance between $\mathbf{x}$ and $\mathbf{x}'$ is less than or equal to b, where b is a positive real number which is small enough to prevent the large modification of $\mathbf{x}$.

The neuron corresponding to the target label y^* is called target neuron. The true probability vector and the predicted probability vector of adversary $\mathbf{x}'$ are denoted by $\mathbf{y}^*$ and $\mathbf{y}'$.

C. Targeted Adversarial Example Generation

AAE. Baluja et al. introduced AAE to generate adversarial examples based on autoencoder [9]. AAE attempts to reconstruct an input vector $\mathbf{x}$ in order to obtain an adversarial example $\mathbf{x}'$. The adversarial example $\mathbf{x}'$ is a reconstructed vector from $\mathbf{x}$, where the prediction label of $\mathbf{x}'$ is the target label y^*. The authors suggested using $||L||_2$ norm to compute the distance between $\mathbf{x}'$ and $\mathbf{x}$. Given a dataset $\mathbf{X}$, their objective function is as follows:

$$loss_{AAE}(\mathbf{X}) = \frac{1}{|\mathbf{X}|} \sum_{\mathbf{x} \in \mathbf{X}} \phi \cdot ||\mathbf{x} - \mathbf{x}'||_2 + ||\mathbf{y}' - r_\alpha(\mathbf{y}, y^*)||_2 \quad (1)$$

, where $r_\alpha(.)$ is a reranking function, ϕ is the weight between the two terms of the objective function, $||\mathbf{y}' - r_\alpha(\mathbf{y}, y^*)||_2$ is the $||L||_2$ distance between $\mathbf{y}'$ and the expected probability vector $r_\alpha(\mathbf{y}, y^*)$.

Box-constrained L-BFGS. Szegedy et al. proposed box-constrained L-BFGS to generate adversarial examples [11] by solving the following constraint:

$$\begin{aligned} &Minimize \quad c \cdot ||\mathbf{x} - \mathbf{x}'||_2 + f(\mathbf{x}', \mathbf{y}^*, \zeta) \\ &subject \quad to \quad \mathbf{x}, \mathbf{x}' \in [lower, upper]^d \end{aligned}$$

, where c is the weight between the distance $||\mathbf{x} - \mathbf{x}'||_2$ and the objective function $f(\mathbf{x}', \mathbf{y}^*, \zeta)$. $upper$ and $lower$ are the upper bound and lower bound of input features, respectively.

Carlini-Wagner $||L||_2$ Attack. The author suggested using the following objective function, denoted by $f_6(.)$:

$$f_6(\mathbf{x}') = max(\overbrace{max\{Z(\mathbf{x}')_i : \forall i \neq y^*\}}^{\text{(i)}} - \overbrace{Z(\mathbf{x}')_{y^*}}^{\text{(ii)}}, -K)$$

, where $Z(.)$ returns the pre-softmax value of the output layer. Part *(i)* is the largest pre-softmax value except the target neuron. Part *(ii)* is the pre-softmax value of target neuron. The initial adversarial example $\mathbf{x}'$ is equal to $\mathbf{x}$, and then is updated: $\mathbf{x}' = \mathbf{x}' - \eta * \nabla_{\mathbf{x}} f_6(\mathbf{x}')$, where η is learning rate.

III. The proposed method

The proposed method, named AE4DNN, starts with four types of inputs including *(i)* the attacked DNN model (denoted by M), *(ii)* a balanced subset of the training set (denoted by $\mathbf{S} \subseteq \mathbf{X}$), *(iii)* the architecture of autoencoder (denoted by g),

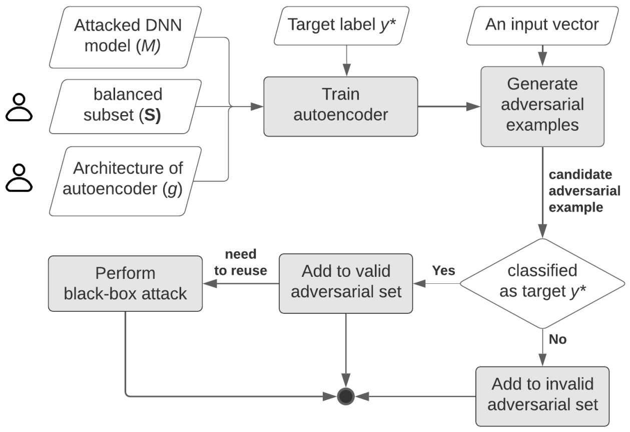
Attacked DNN model (*M*)
Target label *y**
An input vector
balanced subset (**S**)
Train autoencoder
Generate adversarial examples
Architecture of autoencoder (*g*)
candidate adversarial example
need to reuse
Perform black-box attack
Add to valid adversarial set
Yes
classified as target *y**
No
Add to invalid adversarial set

Fig. 1: The main process of AE4DNN.

and *(iv)* the target label (denoted by y^*). The main process of AE4DNN is presented in Fig. 1.

The balanced subset **S** ensures that the class distribution is uniform among the labels except for the target label y^*. The suggestion is that the size of **S** should be large and its characteristics of features should be various enough. There are several main reasons for selecting such a subset. Firstly, the selected autoencoder model can learn all the important features from the samples of **S** among all classes. As a result, for a target label y^*, we just need an autoencoder model to conduct the targeted attack from an input vector belonging to any label. Secondly, using a subset of **X** could reduce the cost of training the autoencoder. If **S** is **X**, in case of large datasets such as Tiny-Images (i.e. 79 million images) [13], it might take high computational cost to train an autoencoder model. This cost might not suitable for the urgent testing of DNN models. For simplicity, AE4DNN selects random samples on the training set **X** to create such a subset **S**.

The architecture of autoencoder g is defined by machine learning testers. The autoencoder should be complex enough to deduce necessary important features of **S** in an intermediate representation and can reconstruct this representation.

After defining the architecture of g and selecting the subset **S**, AE4DNN suggests using the following objective function to train g:

$$loss_g(\mathbf{S}) = \frac{1}{|\mathbf{S}|}\sum_{\mathbf{x}\in\mathbf{S}} \frac{1-\beta}{d}\cdot||\mathbf{x}-g(\mathbf{x}))||_2^2 + \beta\cdot ce(\mathbf{y}_{g(\mathbf{x})}, \mathbf{y}^*) \quad (2)$$

, where $ce(.)$ denotes the cross-entropy function, d is the number of features, $g(\mathbf{x})$ is the reconstructed vector generated from the input vector **x**, $\mathbf{y}_{g(\mathbf{x})}$ is the predicted probability vector of $g(\mathbf{x})$. Two parameters $\frac{1-\beta}{d}$ and β are the weights between the $||L||_2$ distance and the cross-entropy, respectively.

In order to produce adversarial examples which are classified as target label y^*, AE4DNN uses cross-entropy function in Formula (2) to compare the difference between the true probability vectors $\mathbf{y}^*$ and predicted probability vectors $\mathbf{y}_{g(\mathbf{x})}$. This differs from AAE, which applied $||L||_2$ distance instead. The main reason for using cross-entropy is that AE4DNN would have a better generalization and transferability than AAE. Using cross-entropy makes the generated adversarial examples are less dependent on the predicted probability of input vector.

In the Formula (2), choosing the best value of β is a challenging issue. Choosing a good value of β helps to produce adversarial examples which are modified slightly from its input vectors and classified as target label. To address this issue, AE4DNN suggests using an equation to specify a good range of value of β. Specifically, choosing β should ensure that the value of the term $\frac{1-\beta}{d}\cdot||\mathbf{x}-g(\mathbf{x}))||_2^2$ is greater than the value of the term $\beta\cdot cross\_entropy(\mathbf{y}_{g(\mathbf{x})}, \mathbf{y}^*)$. In other words, the autoencoder must satisfy that the reconstructed vector and its input vector should be close to each other, which is considered the most important criterion. The less important criterion is that the prediction of the reconstructed vector is the target label y^*. Given an autoencoder g after initializing its weights, the value of β should be chosen as follows:

$$\beta <= \frac{\sum_{\mathbf{x}\in\mathbf{S}}\frac{1}{d}\cdot||\mathbf{x}-g(\mathbf{x}))||_2^2}{\sum_{\mathbf{x}\in\mathbf{S}}(\frac{1}{d}\cdot||\mathbf{x}-g(\mathbf{x}))||_2^2 + cross\_entropy(\mathbf{y}_{g(\mathbf{x})}, \mathbf{y}^*))} \quad (3)$$

After training an autoencoder, the adversarial example generation from an input vector by using this autoencoder is as follows. The input vector can be on the remaining subset of **X** or a new input vector. Initially, the autoencoder tries to reconstruct this input vector, which results in a candidate adversarial example. This candidate adversarial example is then checked if it is valid. If the candidate adversarial example is classified as the target label y^*, this example is a valid adversarial example, and otherwise, then the algorithm terminates. Note that each target label requires that a corresponding autoencoder is trained. If the attack performs with k labels, the number of autoencoder models is k models, in which each model is used to attack a target label.

IV. Experiment

In order to show the effectiveness of AE4DNN, the experiment addresses the following research questions:

- *In the objective function of AE4DNN, is the proposed formula (3) useful to provide a good range of weight β?* (denoted by RQ1)
- *Compared to AAE, is AE4DNN more effective in dealing with a set of new input vectors? (generalization ability)* (RQ2)
- *Compared to AAE, is AE4DNN more effective in attacking other models? (transferable ability)* (RQ3)
- *Compared to Carlini-Wagner $||L||_2$ attack and box-constrained L-BFGS, does AE4DNN mitigate the trade-off between the quality of adversarial examples and its computational cost?* (RQ4)

The first research question aims to show that the proposed formula (3) would help to specify the good value of β

in formula (2). Based on the answer of the first research question, the second and third research questions show the generalization ability and transferable ability of AE4DNN. The compared method in these two questions is AAE, which is based on autoencoder as AE4DNN. The final research question shows how the proposed method mitigates the trade-off between the quality of adversarial example and its computational cost, which still remains in Carlini-Wagner $||L||_2$ attack and box-constrained L-BFGS. To answer these above questions, a tool has been implemented in python [1]. The experiment is carried out on Google Colab[2].

A. Dataset and attacked model

Dataset MNIST. The research chooses MNIST [12] which is a popular publicly-available dataset for evaluation. The training set contains 50,000 samples. The test set has 10,000 samples. Each sample on the dataset is an image with 28 pixels in width and 28 pixels in height. The value of each pixel is in range of 0 and 255, which indicates the lightness or darkness of that pixel. Adversarial example in this experiment is called *adversarial image* for simplicity.

Attacked model. MNIST dataset is trained with the CNN architecture described in [14]. The main reason why this experiment chooses this architecture is that this CNN architecture is proven the robustness against adversarial attack. The CNN model achieves 99.5% accuracy on MNIST, comparable to the state-of-the-art models.

The architecture of autoencoder. It is challenging to choose the best architecture of autoencoder which attacks well with all DNN models in general and CNN models in particular. Instead, the architecture of the autoencoder should be defined manually based on the expertise of machine learning attackers. In this experiment, the architecture of autoencoder is (input -> conv2d -> maxpooling2d -> conv2d -> maxpooling2d -> conv2d -> upsampling2d -> conv2d -> upsampling2d -> conv2d). This selected autoencoder is a CNN model, which is complex enough to reconstruct an output image from an original image while ensuring that this output image tends to be adversarial.

Each target label requires a separate training process. Therefore, if machine learning testers want to perform targeted attacks for k labels, these testers need to train k autoencoders. For simplicity, this research performs a targeted attack with a target label 7, but other target labels still give the same results.

Rather than training on the whole dataset MNIST, the experiment selects 1,000 first images on MNIST, denoted by $\mathbf{S}$, to generate adversarial examples. There are two reasons leading to this decision. Firstly, the statistical information on $\mathbf{S}$ satisfies that the numbers of images belong to k labels are nearly equivalent. Therefore, the autoencoder might learn enough necessary transformation to convert an image classified as any label to the target label. Secondly, the experiment aims to show the generalization of the trained autoencoder to generate new adversarial examples. Assume that the autoencoder is only trained on $\mathbf{S}$, which is a balanced subset of the training set $\mathbf{X}$, it still shows its usefulness to generate adversarial examples from new input images which are never learnt by the autoencoder.

[1]https://github.com/ducanhnguyen/mydeepconcolic.git

[2]https://colab.research.google.com/

The proposed autoencoder is trained up to 400 epochs with $batch\_size = 256$. We applied early stopping strategy to stop the training process when there is no decrease in the loss over a number of continuous epochs. After the training process, each image of $\mathbf{S}$ would be put into this autoencoder to generate the corresponding candidate adversarial example. This candidate adversarial example is then predicted by the attacked CNN model to detect if it is a valid adversarial example.

B. Comparison

Answer for RQ1. The conclusion of this research question is that the proposed formula (3) might reduce the effort of β selection. Initially, given the subset $\mathbf{S}$, we observed that with different initialization of weights, the value of β is not changed too much (i.e. 0.015 +/- 0.002). This means that the value of β might be less dependent on the weight initialization. Therefore, for simplicity, the upper bound of β in the objective function of AE4DNN is chosen 0.015. If β = 0.015, the importance of the distance of the adversarial examples and the input vectors is the same as the importance of cross-entropy between the original predictions and the expected predictions.

The experiment tries with different values of β in the range of [0, 0.015]. It is expected that this range might contain good values of β, which leads to the high quality of adversarial examples. The quality of adversarial examples is defined as the average of $||L||_2$ distance and the number of adversarial examples. A value of β is considered *good* when the average of $||L||_2$ distance is small and the number of adversarial examples is large enough. Empirically, the average of $||L||_2$ distance should be less than 6.5, which ensures a slight modification of input vectors.

TABLE I: The average $||L||_2$ distance and the corresponding number of adversarial examples with different values of β in AE4DNN. Good values of β are marked in bold.

β	0.0005	**0.001**	**0.002**	**0.003**	**0.004**	**0.005**	0.006
Avg $\|\|L\|\|_2$	3.2	4.56	4.71	4.98	6.47	6.34	7.01
#adv	12	62	75	101	492	661	482

Table I shows different values of β, in which good values of β are marked in bold. In this table, $\beta = 0.0005$ is not good because the number of adversarial examples is too small (i.e. 12 out of 1,000). In addition, $\beta = 0.006$ is also ignored since the generated adversarial examples contain much noise (i.e. the average of $||L||_2$ distance is greater than 6.5), which easily make the attacked model predict incorrectly.

In contrast, if machine learning testers have no idea about a good range of value of β, they usually use the strategy try-and-check until they find out such a good range. This strategy might be time-consuming.

Answer for RQ2. This part evaluates the generalization of AE4DNN to generate adversarial examples from a new set of input vectors compared to AAE. A new input vector is not used to train these autoencoders. The experiment shows that AE4DNN is able to produce a better rate of generalization.

The methods Carnili-Wagner $||L||_2$ attack and box-constrained L-BFGS are not included in this comparison. The main reason is that these methods are not designed to generate new adversarial examples from new input vectors with low computational cost. Instead, these methods generate adversarial examples from the beginning and each input vector requires a separate training process, which lead to high computational cost.

In order to answer this research question, we need to specify good values of ϕ in the objective function of AAE firstly, then use these results to compare with AE4DNN. Specifically, we run AAE with different values of ϕ. Table II illustrates the result of AAE in terms of average $||L||_2$ distance and its corresponding number of adversarial examples with different values of ϕ. The range [0, 0.002) and (0.02, $+\infty$] are ignored because these ranges do not produce a good set of adversarial examples.

TABLE II: The average $||L||_2$ distance and the corresponding number of adversarial examples with different values of ϕ (AAE).

ϕ	0.02	0.01	0.005	0.003	0.002
avg $\|\|L\|\|_2$	4	4.81	6.2	6.8	8.8
#adv	47	66	340	522	696

There is a problem that AE4DNN and AAE use different configuration (i.e. β in AE4DNN and ϕ in AAE) to generate adversarial examples. Therefore, in order to make a comparison, the experiment grouped equivalent configurations by analyzing Table I and Table II, in which a configuration is defined as a pair (β, ϕ). A pair of configuration (β, ϕ) is equivalent when AE4DNN and AAE produce approximate values of $||L||_2$ distance. There are five pairs of configuration satisfying this requirement, including $A = (0.001, 0.01)$, $B = (0.002, 0.01)$, $C = (0.003, 0.01)$, $D = (0.004, 0.005)$, and $E = (0.005, 0.005)$.

This experiment then used three sets of new input vectors with the size 10,000 images (denoted by *10k-attack*), 20,000 images (*20k-attack*), and 40,000 images (*40k-attack*) for evaluation. Because the result of *10k-attack*, *20k-attack*, and *40k-attack* are nearly the same, we compute the average of comparable criterion for simplicity, which is shown in Table III. Here, *average adversarial rate* $\in [0, 1]$ is the average proportion of the number of adversarial images to the number of input images (i.e. higher value is better). As can be seen, AE4DNN produces a better average adversarial rate compared to AAE for all pairs of configuration because of the usage of cross-entropy in Formula (2).

Answer for RQ3. This part examines the transferable ability of AE4DNN compared to AAE. The procedure of transferable attack is as follows. Initially, adversarial examples would be produced by attacking the trained CNN model previously with 1,000 first images of MNIST. After that, these adversarial examples are put into different DNN models including VGG-13 model, VGG-16 model, LeNet-5 model, and AlexNet model. If an adversarial example makes the new model predict as the target label y^*, this adversarial example could be used to attack this new model. The detail of comparison is shown in Table V. As can be seen, AE4DNN. produces better transferability rate for all pair of configurations, compared to AAE.

TABLE III: The comparison between AE4DNN and AAE in terms of generalization. Target label is 7. Better values are marked in bold. The total time to perform 10k-attack, 20k-attack, and 40k-attack are approximate to 1.6 seconds, 3.1 seconds, and 6.3 second, respectively. These attacks do not need to train the autoencoder.

Config	Average $\|\|\|L\|\|_2$		Average adversarial rate (%)	
	AE4DNN	AAE	AE4DNN	AAE
A	**4.65**	4.76	**7.5**	7.1
B	**4.69**	4.76	**8.5**	7.1
C	4.99	**4.76**	**11.7**	7.1
D	6.49	**6.28**	**53.5**	39
E	6.38	**6.28**	**67.7**	39

Answer for RQ4. This part shows how AE4DNN mitigates the trade-off between the quality of adversarial example and its computational cost, compared to Carlini-Wagner $||L||_2$ attack and box-constrained L-BFGS. For each method, we empirically choose good ranges of configuration to make a comparison. By ignoring the unusual adversarial examples, Table IV gives a summary comparison with different configurations. Some numbers rounded to increase the readability of this table. *Configuration* represents the range of parameters which is used for the comparison. *Min/Max* $||L||_2$ *distances* are the minimum distance and the maximum distance between an input image and its corresponding adversarial example, respectively. *Time* is the total time to generate adversarial examples for a specific configuration. For AE4DNN and AAE, *time* is computed from the two steps including *(i)* the training process of an autoencoder and (ii) adversarial example generation from the trained autoencoder. As can be seen, AE4DNN works better than Carlini-Wagner $||L||_2$ attack in terms of min/max $||L||_2$ distances and time. Although AE4DNN and box-constrained L-BFGS have approximate min/max $||L||_2$ distances, AE4DNN spent less computational cost to generate adversarial examples (around 1 minute).

TABLE IV: The comparison between AE4DNN with different state-of-the-art adversarial example generation techniques. The experiment is carried out on the subset **S** of the training set of MNIST. All unusual adversarial examples are ignored. We re-implement box-constrained L-BFGS and re-run Carlini-Wagner $||L||_2$ attack[3].

Criteria	AE4DNN	box-constrained L-BFGS	Carlini-Wagner $\|\|L\|\|_2$ attack
Configuration	[0.001, 0.005]	[0.001, 0.0035], iter = 20	-
Min $\|\|L\|\|_2$	3.85	2.34	13.43
Max $\|\|L\|\|_2$	6.34	7.4	15.08
Time (minute)	~1	~3	~30

V. Related Works

Currently, some works have been proposed for testing DNN models by several research groups. Focusing only on the most recent and closest ones, we can refer to [7], [9]–[11], [15].

TABLE V: The transferable rate between AE4DNN and AAE with different DNN models. Better values are marked in bold.

Config	VGG-13 (%)		VGG-16 (%)		LeNet-5 (%)		AlexNet (%)	
	AE4DNN	AAE	AE4DNN	AAE	AE4DNN	AAE	AE4DNN	AAE
A	**6.3**	4.9	**3.7**	3.6	**1.2**	0.9	**7.3**	6.1
B	**7.4**	4.9	**6**	3.6	**1.2**	0.9	**8**	6.1
C	**31.8**	4.9	**27.7**	3.6	**10**	0.9	**20.4**	6.1
D	**46.4**	38.5	**47.1**	39.1	**18.1**	6.4	**39.4**	31.1
E	**46.4**	38.5	**47.1**	39.1	**18.1**	6.4	**39.1**	31.1

Goodfellow et al. proposed targeted/untargeted fast gradient sign methods (FGSM) [15] for $||L||_\infty$ distance. Every pixel of an image would be calculated its corresponding derivative firstly. These pixels are then moved by a distance simultaneously which is the multiplication of a constant and derivatives. Their proposal can apply in practice because it has a low computational cost.

Szegedy et al. introduced a targeted attack to generate adversarial examples using box-constrained L-BFGS [11]. The authors defined an alternative objective formula to address the problem of constrained minimization. The alternative objective formula has two terms in which the first represents the $||L||_2$ distance between the original image and its corresponding adversarial example. The second term minimizes the distance between the prediction of the original image and its adversarial example.

Inspired from the work of Szegedy et al. [11], Carlini et al. presented a $||L||_2$ attack [10] to generate adversarial examples. The authors suggested using approximate objective functions to alter the probability comparison of original/target prediction in box-constrained L-BFGS [11]. The alternatives are chosen to be better suited for optimization, so adversarial examples are found out more easily.

Baluja et al. presented an autoencoder-based method to generate adversarial examples [9]. Given a labelled dataset, an autoencoder will be built in such a way that the reconstructed outputs are the adversarial examples. Their proposal is proved to be fast with a small $||L||_2$ distance.

Szegedy et al. proposed l.l. class to generate adversarial examples [7]. This method applies FGSM multiple times with a small step size. At each step, pixel values are clipped to ensure that the images over iterations are valid.

VI. Conclusion

We have presented an autoencoder-based method to generate adversarial examples for DNN models, which is called AE4DNN. The proposed method is compared with the state-of-the-art adversarial example generation methods including box-constrained L-BFGS, Carlini-Wagner $||L||_2$ attack, and AAE. As a result, the comprehensive experiment on MNIST has shown that AE4DNN can obtain a better transferable rate and generalization. Furthermore, AE4DNN is able to mitigate the trade-off between the quality of adversarial examples and the computational cost of state-of-the-art adversarial example generation techniques. Therefore, AE4DNN has a potential advantage to be used in practice, which might reduce the cost of DNN testing significantly. In the future, AE4DNN will be extended to support other distance metrics such as $||L||_0$ and $||L||_\infty$. In addition, the research will make a more comprehensive comparison with other well-known datasets.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[3] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, June 2012, pp. 3642–3649.

[4] D. Ciresan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, “Deep neural networks segment neuronal membranes in electron microscopy images,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 2843–2851. [Online]. Available: http://papers.nips.cc/paper/4741-deep-neural-networks-segment-neuronal-membranes-in-electron-microscopy-images.pdf

[5] A. McCallum and K. Nigam, “A comparison of event models for naive bayes text classification,” in *Learning for Text Categorization: Papers from the 1998 AAAI Workshop*, 1998, pp. 41–48. [Online]. Available: http://www.kamalnigam.com/papers/multinomial-aaaiws98.pdf

[6] H. B. Braiek and F. Khomh, “On testing machine learning programs,” *CoRR*, vol. abs/1812.02257, 2018. [Online]. Available: http://arxiv.org/abs/1812.02257

[7] A. Kurakin, I. J. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” *CoRR*, vol. abs/1607.02533, 2016. [Online]. Available: http://arxiv.org/abs/1607.02533

[8] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” *CoRR*, vol. abs/1705.06640, 2017. [Online]. Available: http://arxiv.org/abs/1705.06640

[9] S. Baluja and I. Fischer, “Adversarial transformation networks: Learning to generate adversarial examples,” 2017.

[10] N. Carlini and D. A. Wagner, “Towards evaluating the robustness of neural networks,” *CoRR*, vol. abs/1608.04644, 2016. [Online]. Available: http://arxiv.org/abs/1608.04644

[11] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” 2014.

[12] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[13] A. Torralba, R. Fergus, and W. T. Freeman, “80 million tiny images: A large data set for nonparametric object and scene recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[14] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, “Distillation as a defense to adversarial perturbations against deep neural networks,” *CoRR*, vol. abs/1511.04508, 2015. [Online]. Available: http://arxiv.org/abs/1511.04508

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” 2015.