Learning to Confuse: Generating Training Time Adversarial Data with Auto-Encoder

Ji Feng[1,2*], Qi-Zhi Cai[1,2*], Zhi-Hua Zhou[1]

[1]*National Key Laboratory for Novel Software Technology*
Nanjing University, Nanjing 210023, China
Email: {fengj/zhouzh}@lamda.nju.edu.cn, caiqizhi@smail.nju.edu.cn
[2]*Sinovation Ventures AI Institute*
Email: fengji@chuangxin.com

Abstract

In this work, we consider one challenging training time attack by modifying training data with bounded perturbation, hoping to manipulate the behavior (both targeted or non-targeted) of any corresponding trained classifier during test time when facing clean samples. To achieve this, we proposed to use an auto-encoder-like network to generate the pertubation on the training data paired with one differentiable system acting as the imaginary victim classifier. The perturbation generator will learn to update its weights by watching the training procedure of the imaginary classifier in order to produce the most harmful and imperceivable noise which in turn will lead the lowest generalization power for the victim classifier. This can be formulated into a non-linear equality constrained optimization problem. Unlike GANs, solving such problem is computationally challenging, we then proposed a simple yet effective procedure to decouple the alternating updates for the two networks for stability. The method proposed in this paper can be easily extended to the label specific setting where the attacker can manipulate the predictions of the victim classifiers according to some predefined rules rather than only making wrong predictions. Experiments on various datasets including CIFAR-10 and a reduced version of ImageNet confirmed the effectiveness of the proposed method and empirical results showed that, such bounded perturbation have good transferability regardless of which classifier the victim is actually using on image data.

---

*Equally contributed.

1. Introduction

How to modify the training data with bounded transferable perturbation that can lead to the largest generalization gap? In other words, we consider the task of adding imperceivable noises to the training data, hoping to maximally confuse any corresponding classifier trained on it by letting it to make the wrong predictions as much as possible when facing clean test data.

To achieve the above motivation, we defined a deep encoder-decoder-like network to generate such perturbations which takes the clean samples as input and outputs the corresponding adversarial noises in the same sample space. Such bounded noises is then added to the training data. Meanwhile, we use an imaginary neural network as the victim classifier, and the goal here is to train both networks simultaneously by letting the autoencoder to update its weights that can cause the lowest test accuracy for the victim classifier. Instead of reconstruction error as learning objective for traditional autoencoders, here we formulated the problem into a non-linear equality constrained optimization problem. Unlike GANs [1], such optimization problem is much harder to solve and a direct implementation of alternating updates will lead to unstable result. Inspired by some common techniques in reinforcement learning such as introducing a separate record tracking network like target-nets to stabilize Q-learning [2], we proposed a similar approach by decoupling the training procedure by introducing the pseudo-update steps when training the autoencoder. By doing so, the optimization procedure is much stable in practice.

A similar setting is data poisoning [3] in the security community but the goal is quite different with this work. The main goal here is to examine the robustness of training data by adding bounded noises in order to reveal some intriguing properties of neural networks, whereas data poisoning focus on the restriction that only few training data is allowed to change. Actually, having full control of training data (instead of changing a few) is a realistic assumption, for instance, in some applications an agent may agree to release some internal data for peer assessment or academic research, but does not like to enable the data receiver to build a model which performs well on real test data; this can be realized by applying such adversarial noises before the data release.

The other contribution of this work is that, such formalization can be easily extended to the label specific case, where one wants to specifically fool the classifier of recognizing one input pattern into a *specifically predefined class*, rather than making a wrong prediction only. Finally, experimental

results showed that, the learned noises is effective and robust to other machine learning models with different structure or even different types such as Random Forest [4] or Support Vector Machine(SVM) [5].

The rest of the paper is organized as follows: First, we will give some more related works followed by the formalization for the proposed problem. Then, the optimization procedure is described followed by a discussion of some variants of the task. Experimental results are presented and finally conclusion and future works are discussed.

2. Related Works

One subject which closely relates to our work is data poisoning. The task of data poisoning dates back to the pre-deep learning times. For instance, there has been some research on poisoning the classical models including SVM [6], Linear Regression [7] and Naive Bayes [8] which basically transform the poisoning task into a convex optimization problem. Poisoning for deep models, however, is a more challenging one. Kon et.al. [9] first proposed the possibility of poisoning deep models via the influence function to derive adversarial training examples. Currently, there have been some popular approaches to data poisoning. For instance, sample specific poisoning aims to manipulate the model's behavior on some particular test samples. [10, 11, 12]. On the other hand, general poison attack aiming to reduce the performance on cleaned whole unseen test set [9, 3]. As explained in the previous section, one of the differences with data poisoning is that the poisoning task mainly focusing on modifying as few samples as possible whereas our work focus on adding bounded noises as small as possible. In addition, our noise adding scheme can be scaled to much larger datasets with good transferability.

Another related subject is adversarial examples or testing time attacks, which refers to the case of presenting malicious testing samples to an already trained classifier. Since the classifier is given and fixed, there is no two-party game involved. Researches showed deep model is very sensitive to such adversarial examples due to the high-dimensionality of the input data and the linearity nature inside deep neural networks [13]. Some recent works showed such adversarial examples also exists in the physical world [14, 15], making it an important security and safety issue when designing high-stakes machine learning systems in an open and dynamic environment. Our work can be regarded as training time analogy of adversarial examples.

There have been some works on explaining the effectiveness of adversarial examples. The work in [16] proposed that it is the linearity inside neural networks that makes the decision boundary vulnerable in high dimensional space. Although beyond the scope of this paper, we tested several hypotheses on explaining the effectiveness of training time adversarial noises.

3. The proposed method

Consider the standard supervised learning procedure for classification where one wants learn the mapping $f_\theta : \mathcal{X} \to \{0,1\}^K$ from data where K is the number of classes being predicted. To learn the optimal parameters θ^*, a loss function such as cross-entropy for classification $\mathcal{L}(f_\theta(x), y) : \mathbb{R}^k \times \mathbb{Z}_+ \to \mathbb{R}_+$ on training data is often defined and empirical risk minimization [17] can thus be applied, that is, one want to minimize the loss function on training data as:

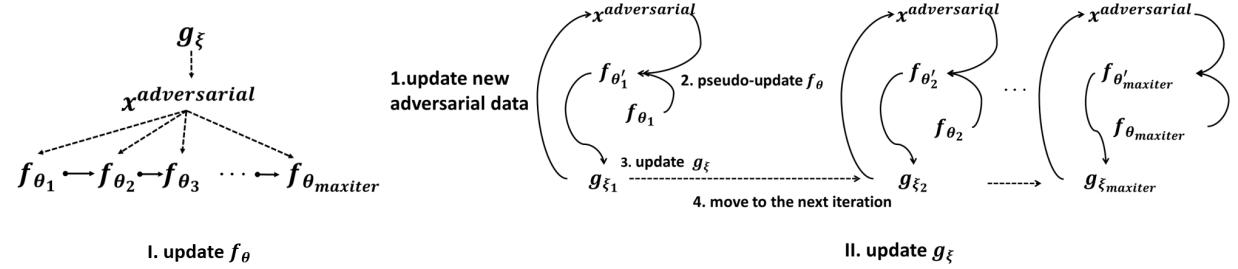$$\theta^* = \arg\min_\theta \sum_{(x,y)\sim\mathcal{D}} [\mathcal{L}(f_\theta(x), y)] \tag{1}$$

Figure 1: An overview for learning to confuse: Decoupling the alternating update for f_θ and g_ξ

4

Algorithm 1: Deep Confuse

Input: Training data $\mathcal{D}$, number of trials T, max iteration for training a classification model $maxiter$, learning rate of classification model α_f, learning rate of the Noise Generator α_g, batch size b

Output: Learned Noise Generator g_ξ

1   $\xi \leftarrow RandomInit()$

2   **for** $t = 1$ **to** T **do**

3      $\theta_0 \leftarrow RandomInit()$

4      $L \leftarrow$ empty list

5   // Update f_θ while keeping g_ξ fixed

6   **for** $i = 0$ **to** $maxiter$ **do**

7   //Sample a mini-batch of training data

8         $(x_i, y_i) \sim \mathcal{D}$

9   L.append$((\theta_i, x_i, y_i))$

10        $x_i^{adversarial} \leftarrow x_i + g_\xi(x_i)$

11   // Update model f_θ by SGD

12        $\theta_{i+1} \leftarrow \theta_i - \alpha_f \nabla_{\theta_i} \mathcal{L}(f_{\theta_i}(x_i^{adversarial}), y_i)$

13   **end**

14   // update g_ξ via a pseudo-update of f_θ

15   **for** $i = 0$ **to** $maxiter$ **do**

16        $(\theta_i, x_i, y_i) \leftarrow L[i]$

17   //pseudo-update f_θ over the current adversarial data

18        $\theta' \leftarrow \theta_i - \alpha_f \nabla_{\theta_i} \mathcal{L}(f_{\theta_i}(x_i + g_\xi(x_i)), y_i)$

19   // Update g_ξ over clean data

20        $\xi \leftarrow \xi + \alpha_g \nabla_\xi \mathcal{L}(f_{\theta'}(x), y)$

21   **end**

22 **end**

23 **return** g_ξ

When f_θ is a differentiable system such as neural networks, stochastic gradient descent (SGD) [18]

or its variants can be applied by updating θ via gradient descent

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(f_\theta(x), y), \tag{2}$$

where α refers to the learning rate.

The goal for this work is to perturb the training data by adding artificially imperceivable noise such that during testing time, the classifier's behavior will be dramatically different on the clean test-set.

To formulate this, we first define a noise generator $g_\xi : \mathcal{X} \to \mathcal{X}$ which takes one training sample x in $\mathcal{X}$ and transform it into an imperceivable noise pattern in the same space $\mathcal{X}$. For image data, such constraint can be formulated as:

$$\forall x, \|g_\xi(x)\|_\infty \leq \epsilon \tag{3}$$

Here, the ϵ controls the perturbation strength which is a common practice in adversarial settings [13]. In this work, we choose the noise generator g_ξ to be an encoder-decoder neural network and the activation for the final layer is defined to be:$\epsilon \cdot (\tanh(\cdot))$ to facilitate the constraint (3).

With the above motivation and notations, we can then formalize the task into the following optimization problem as:

$$
\begin{aligned}
\max_{\xi} \quad & \sum_{(x,y)\sim\mathcal{D}} [\mathcal{L}(f_{\theta^*(\xi)}(x), y)], \\
s.t. \quad & \theta^*(\xi) = \arg\min_{\theta} \sum_{(x,y)\sim\mathcal{D}} [\mathcal{L}(f_\theta(x + g_\xi(x)), y)]
\end{aligned}
\tag{4}
$$

In other words, every possible configuration ξ is paired with one classifier $f_{\theta^*(\xi)}$ trained on the corresponding modified data, the goal here is to find a noise generator g_{ξ^*} such that the paired classifier $f_{\theta^*(\xi^*)}$ to have the worst performance on the cleaned test set, compared with all the other possible ξ.

This non-convex optimization problem is challenging, especially due to the nonlinear equality constraint. Here we propose an alternating update procedure using some commonly accepted

6

tricks in reinforcement learning for stability [2] which is simple yet effective in practice.

Algorithm 2: Mem-Efficient Deep Confuse

Input: Training data $\mathcal{D}$, number of trials T, max iteration for training a classification model $maxiter$, learning rate of classification model α_f, learning rate of the Noise Generator α_g, batch size b

Output: Learned Noise Generator g_ξ

1   $\xi \leftarrow RandomInit()$

2   $g'_\xi \leftarrow g_\xi.copy()$

3   **for** $t = 1$ **to** T **do**

4      $\theta_0 \leftarrow RandomInit()$

5   **for** $i = 0$ **to** $maxiter$ **do**

6   //Sample a mini-batch

7         $(x_i, y_i) \sim \mathcal{D}$

8   // Update g'_ξ using current f_θ

9         $\theta' \leftarrow \theta_i - \alpha_f \nabla_{\theta_i} \mathcal{L}(f_{\theta_i}(x_i + g'_\xi(x_i)), y_i)$

10        $\xi' \leftarrow \xi' + \alpha_g \nabla_{\xi'} \mathcal{L}(f_{\theta'}(x), y)$

11   // Update f_θ by SGD

12        $x_i^{adversarial} \leftarrow x_i + g_\xi(x_i)$

13        $\theta_{i+1} \leftarrow \theta_i - \alpha_f \nabla_{\theta_i} \mathcal{L}(f_{\theta_i}(x_i^{adversarial}), y_i)$

14   **end**

15      $g_\xi \leftarrow g'_\xi$

16   **end**

17   **return** g_ξ

First, since we are assuming f_θ and g_ξ to be neural networks, the equality constraint can be relaxed into

$$\theta_i = \theta_{i-1} - \alpha \cdot \nabla_{\theta_{i-1}} \mathcal{L}(f_{\theta_{i-1}}(x + g_\xi(x)), y) \tag{5}$$

where i is the index for SGD updates.

Second, the basic idea is to alternatively update f_θ over adversarial training data via gradient descent and update g_ξ over clean data via gradient ascent. The main problem is that, if we directly

using this alternating approach, both networks f_θ and g_ξ won't converge in practice. To stabilize this process, we propose to update f_θ over the adversarial training data first, while collecting the update trajectories for f_θ, then, based on such trajectories, we update the adversarial training data as well as g_ξ by calculating the pseudo-update for f_θ at each time step. Such whole procedure is repeated T trials until convergence. The detailed procedure is illustrated in Algorithm 1 and Figure 1.

Finally, we introduce one more modification for efficiency. Notice that storing the whole trajectory of the gradient updates when training f_θ is memory inefficient. To avoid directly storing such information, during each trial of training, we can create a copy of g_ξ as g_ξ' and let g_ξ' to alternatively update with f_θ, then copy the parameters back to g_ξ. By doing so, we can merge the two loops within each trial into a single one and doesn't need to store the gradients at all. The detailed procedure is illustrated in Algorithm 2.

4. Label Specific Adversaries

In this section, we give a brief introduction of how to transfer our settings to the label specific scenarios. The goal for label specific adversaries is that the adversary not only wants the classifier to make the wrong predictions but also want the classifier's predictions specifically according to some pre-defined rules. For instance, the attacker wants the classifier to wrongly recognize the pattern from class A specifically to Class B (thus not to Class C). To achieve this, denote $\eta : \mathbb{Z}_+ \to \mathbb{Z}_+$ as a predefined label transformation function which maps one label to another. Here η is pre-defined by the attacker, and it transforms a label index into another different label index. Such label specific adversary can thus be formalized into:

$$
\begin{aligned}
&\min_\xi \quad \sum_{(x,y)\sim\mathcal{D}} [\mathcal{L}(f_{\theta^*(\xi)}(x), \eta(y))], \\
&s.t. \quad \theta^*(\xi) = \arg\min_\theta \sum_{(x,y)\sim\mathcal{D}} \mathcal{L}(f_\theta(x_i + g_\xi(x_i)), y_i)
\end{aligned}
\tag{6}
$$

It is easy to show that optimizing the above problem is nearly identical with the procedure described in Algorithm 2. The only thing needed to be changed is to replace the gradient ascent

8

into gradient decent in line 10 in Algorithm 2 and replace $\eta(y)$ to y in the same line while keeping others unchanged.

5. Experiment

(a) 2-Class ImageNet
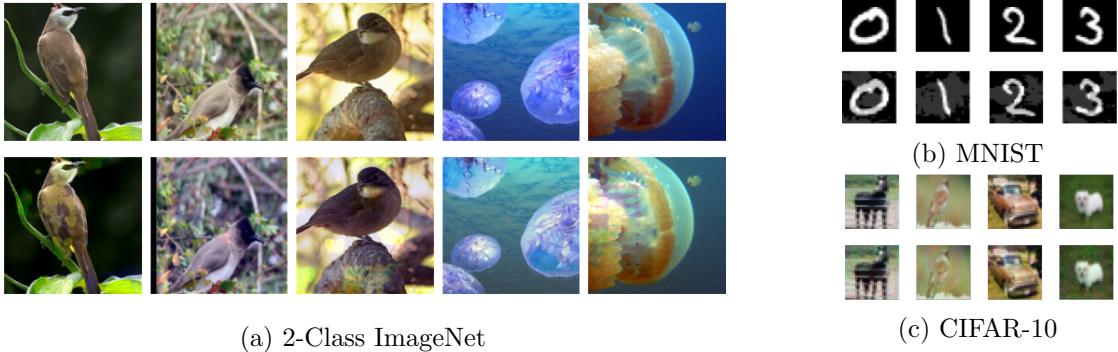

(b) MNIST

(c) CIFAR-10

Figure 2: First Rows: original training samples. Second rows: adversarial training samples.

To validate the effectiveness of our method, we used the classical MNIST [19], CIFAR-10 [20] for multi-classification and a subset of ImageNet [21] for 2-class classification. Concretely, we used a subset of ImageNet (bulbul v.s. jellyfish) consists of 2,600 colored images with size $224{\times}224{\times}3$ for training and 100 colored images for testing. Random samples for the adversarial training data is illustrated in Figure 2.

The classifier f_θ during training we used for MNIST is a simple Convolutional Network with 2 convolution layers having 20 and 50 channels respectively, followed by a fully-connected layer consists of 500 hidden units. For the 2-class ImageNet and CIFAR-10, we used f_θ to be a CNN with 5 convolution layers having 32,64,128,128 and 128 channels respectively, each convolution layer is followed by a $2{\times}2$ pooling operations. Both classifiers used ReLU as activation and the kernel size is set to be $3{\times}3$. Cross-entropy is used for loss function whereas the learning rate and batch size for the classifiers f_θ are set to be 0.01 and 64 for MNIST and CIFAR-10 and 0.1 and 32 for ImageNet. The number of trials T is set to be 500 for both cases.

The noise generator g_ξ for MNIST and ImageNet consists of an encoder-decoder structure where each encoder/decoder has 4 4x4 convolution layers with channel numbers 16,32,64,128 respectively. For CIFAR-10, we use a U-Net [22] which has larger model capacity. The learning rate for the noise generator g_ξ is set to be 10^{-4} via Adam [23].

9

5.1. Performance Evaluation of Training Time Adversary

Using the model configurations described above, we trained the noise generator g_ξ and its corresponding classifier f_θ with perturbation constraint ϵ to be 0.3, 0.1, 0.032, for MNIST, ImageNet and CIFAR-10, respectively. The classification results are summarized in Table 1. Each experiment is repeated 10 times.
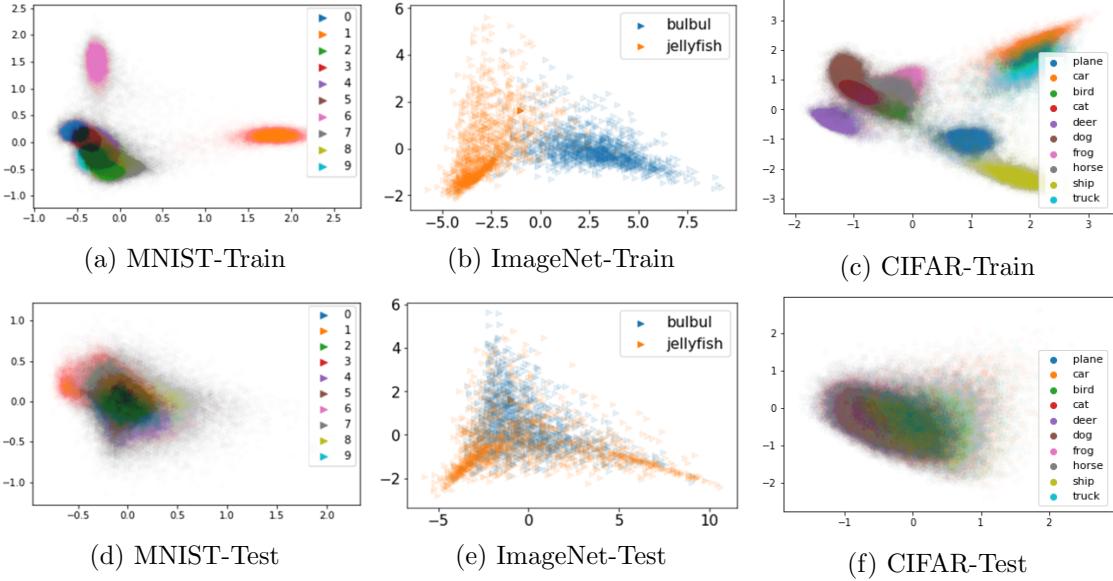

(a) MNIST-Train (b) ImageNet-Train (c) CIFAR-Train

(d) MNIST-Test (e) ImageNet-Test (f) CIFAR-Test

Figure 3: First Row: Deep features of the adversarial training data. Second Row: Deep features of the cleaned test data.

Table 1: Test accuracy (mean±std) when the classifier is trained on the original clean training set and the adversarial training set,respectively.

	MNIST	ImageNet	CIFAR-10
Clean Data	99.32 ± 0.05	88.5 ± 2.32	77.28 ± 0.17
Adversarial Data	0.25 ± 0.04	54.2 ± 11.19	28.77 ± 2.80

When trained on the adversarial datasets, the test accuracy dramatically dropped to only 0.25±0.04, 54.2 ± 11.19 and 28.77 ± 2.80, a clear evidence of the effectiveness for the proposed method.

We also visualized the activation of the final hidden layers of f_θs trained on the adversarial training sets in Figure 3. Concretely, we fit a PCA [24] model on the final hidden layer's output for each f_θ on the adversarial training data, then using the same projection model, we projected the clean data into the same space. It can be shown that the classifier trained on the adversarial data cannot differentiate the clean samples.

It is interesting to know how does the perturbation constraint ϵ affect the performance in terms of both accuracy and visual appearance. Concretely, on MNIST dataset, we varied ϵ from 0 (no modification) to 0.3, with a step size of 0.05 while keeping other configurations the same and the results are illustrated in Figure 4. Test accuracy refers to the corresponding model performance trained on the different adversarial training data with different ϵ. From the experimental result, we observed a sudden drop in performance when ϵ exceeds 0.15. Although beyond the scope of this work, we conjecture this result is related or somewhat consistent with a similar theoretical guarantee for the robust error bound when ϵ is 0.10 [25].

Finally, We examined the results when the training data is partially modified. Concretely, under different perturbation constraint, we varied the percentage of adversaries in the training data while keeping other configurations the same. The results are demonstrated in Figure 5. Random flip refers to the case when one randomly flip the labels in the training data.
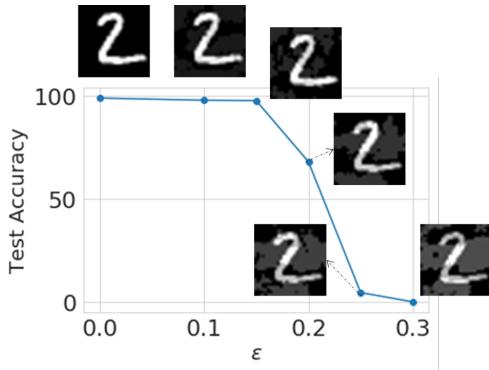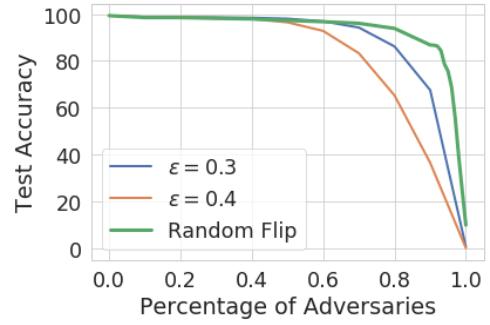

Figure 4: Effect of varying ϵ.

Figure 5: Varying the ratio of adversaries under different ϵ.

5.2. Evaluation of Transferability

In a more realistic setting, it is important to know the performance when we use a different classifier. Concretely, denote the original conv-net f_θ been used during training as $\text{CNN}_{\text{original}}$. After the adversarial data is obtained, we then train several different classifiers on the same adversarial data and evaluate their performance on the clean test set.

For MNIST, we doubled/halved all the channels/hidden units and denote the model as $\text{CNN}_{\text{large}}$ and $\text{CNN}_{\text{small}}$ accordingly. In addition, we also trained a standard Random Forest [4] with 300 trees and a SVM [5] using RBF kernels with kernel coefficient equal to 0.01. The experimental

results are summarized in Figure 6.Blue histograms correspond to the test performance trained on the clean dataset, whereas orange histograms correspond to the test performance trained on the adversarial dataset. From the experimental results, it can be shown that the adversarial noises produced by g_ξ are general enough such that even non-NN classifiers as random forest and SVM are also vulnerable and produce poor results as expected.
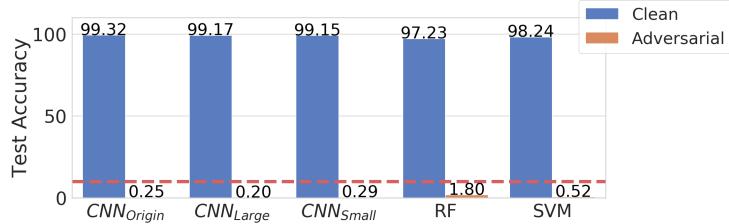

Figure 6: Test performance when using different classifiers. The horizontal red line indicates random guess accuracy.

For CIFAR-10 and ImageNet, we tried a variety of conv-nets including VGG [26], ResNet [27] and DenseNet [28] with different layers, and evaluate the performance accordingly. The results are summarized in Figure 7. Again, good transferability of the adversarial noise is observed.
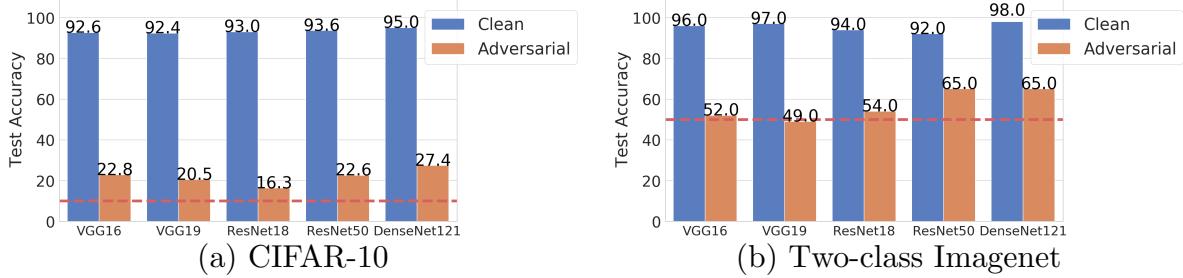

(a) CIFAR-10   (b) Two-class Imagenet

Figure 7: Test performance when using different model architectures.The horizontal red line indicates random guess accuracy.

5.3. The Generalization Gap and Linear Hypothesis

To fully illustrate the generalization gap caused by the adversarials, after we obtained the adversarial training data, we retrained 3 conv-nets (one for each data-sets) having the same architecture as f_θ and plotted the training curves as illustrated in Figure 8. A clear generalization gap between training and testing is observed. We conjecture the deep model tends to *over-fits towards the training noises* $g_\xi(x)$.

(a) MNIST.　　　　(b) 2-class ImageNet.　　　　(c) CIFAR-10.

Figure 8: Learning curves for f_θ

To validate our conjecture, we measured the predictive accuracy between the true label and the *predictions $f_\theta(g_\xi(x))$ taking only adversarial noises as inputs.* The results are summarized in Table 2. Notice 95.15%, 93.00% and 72.98% test accuracy is obtained on the test set.

This interesting result confirmed the conjecture that the model does over-fit to the noises. Here we give one possible explanation. We hypothesize that it is the linearity inside deep models that make the adversarial effective. In other words, $f_\theta(g_\xi(x)$ contributes most when minimizing $\mathcal{L}(f_\theta(x + g_\xi(x)), y)$. This result is deeply related and consistent with the results from adversarial examples [13] and the memorization property for DNNs [29].
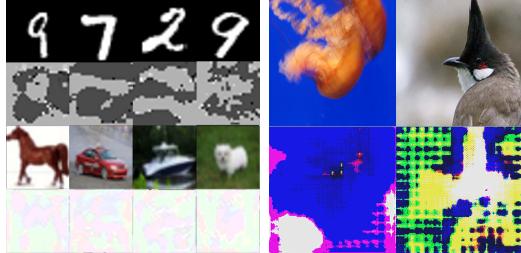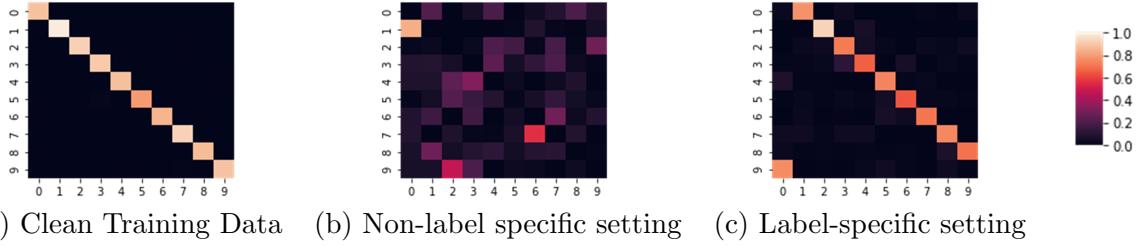

Figure 9: Clean samples and their corresponding adversarial noises for MNIST, CIFAR-10 and ImageNet

Table 2: Prediction accuracy taking **only noises as inputs**. That is, the accuracy between the true label and $f_\theta(g_\xi(x))$ where x is the clean sample.

	Noise$_{\text{train}}$	Noise$_{\text{test}}$
MNIST	95.62	95.15
ImageNet	88.87	93.00
CIFAR-10	78.57	72.98

5.4. Label Specific Adversaries

To validate the effectiveness in label specific adversarial setting, without loss of generalizability, here we shift the predictions by one. For MNIST dataset, we want the classifier trained on the adversarial data to predict the test samples from class 1 *specifically* to class 2, and class 2 to class 3 ... and class 9 to class 0. Using the method described in section 4, we trained the corresponding noise generator and evaluated the corresponding CNN on the test set, as illustrated in Figure 10.

13

(a) Clean Training Data   (b) Non-label specific setting   (c) Label-specific setting

Figure 10: The confusion matrices on test set under different scenarios for MNIST dataset. They summarized the test performance of classifier trained on (a) clean training data (b) Non-label specific setting and (c) label-specific setting.

Compared with the test accuracy (0.25 ± 0.04) in the non-label specific setting, the test accuracy also dropped to 1.48 ± 0.21, in addition, the success rate for targeting the desired specific label increased from 0.00 to 79.7 ± 0.38. Such results gave positive supports for the effectiveness in label specific adversarial setting. Notice this is only a side-product of the proposed method to show the formulation can be easily modified to achieve some more user-specific tasks.

6. Conclusion

In this work, we propose a general framework for generating training time adversarial data by letting an auto-encoder watch and move against an imaginary victim classifier. We further proposed a simple yet effective training scheme to train both networks by decoupling the alternating update procedure for stability. Experiments on image data confirmed the effectiveness of the proposed method, in particular, such adversarial data is still effective even to use a different classifier, making it more useful in a realistic setting.

Theoretical analysis or some more improvements for the optimization procedure is planned as future works. In addition, it is interesting to design adversarially robustness classifiers against this scheme.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *ACM Workshop on Artificial Intelligence and Security*, pages 27–38, 2017.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[6] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, pages 1467–1474, 2012.

[7] Matthew Jagielski, Alina Oprea, Battista Biggio, Chang Liu, Cristina Nita-Rotaru, and Bo Li. Manipulating machine learning: Poisoning attacks and countermeasures for regression learning. In *IEEE S & P*, pages 19–35, 2018.

[8] Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. Exploiting machine learning to subvert your spam filter. In *Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 7:1–7:9, 2008.

[9] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *ICML*, pages 1885–1894, 2017.

[10] Ali Shafahi, W. Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *NIPS*, pages 6106–6116, 2018.

[11] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[12] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[13] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

[14] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634, 2018.

[15] Anish Athalye and Ilya Sutskever. Synthesizing robust adversarial examples. In *ICML*, pages 284–293, 2018.

[16] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint*, 2013.

[17] Vladimir Vapnik. Principles of risk minimization for learning theory. In *NIPS*, pages 831–838, 1992.

[18] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*, pages 177–186, 2010.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[23] Diederik P Kingma and Jimmy Lei Ba. Adam: Amethod for stochastic optimization. In *ICLR*, 2014.

[24] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[25] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *ICML*, pages 5283–5292, 2018.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[28] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proc. of CVPR*, pages 2261–2269, 2017.

[29] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *ICLR*, 2016.