6.883 Mid-Project Report: Adversarial Generation and Perturbation Elimination with GANs: Re-implement and Evaluate APE-GAN, Defense-GAN and AdvGAN

Mucong Ding, Sirui Lu, Zhiwei Ding

This is the mid-project report of 6.883 Spring 2018. The topic of our project is adversarial generation and perturbation elimination with GANs. The first half of our research mainly focus on re-implement and evaluate the three GAN-based adversarial attack and defense models: APE-GAN [1], Defense-GAN [2], and AdvGAN [3]. More specifically, we first re-implement each model either entirely from scratch (AdvGAN) or from the released code (APE-GAN and Defense-GAN (a partial implementation)). After that, we try to reproduce important results in the three papers and ensure they are reliable. Lastly, we evaluate their performance by some state-of-arts attacks and defenses. This work-in-progress report summarizes all the implementation details and evaluation results so far, together with the research question and obstacles. We believe the working experiences on these three models can help us move towards our final objective: trying to improve the GAN-based adversarial attacks and defenses by combining them and making use of their advantages.

I. Introduction

Following the proposed plan, we reimplement and evaluate the three GAN-based attack and defense models during the first half of the project. The working experiences on the three models is critical to our final objective: improving GAN-based attacks and defenses. It also helps us crystallize our ideas and formulate a detailed and concrete research plan for the second half of project.

This rest of this report is structured as follows:

- Section. II: APE-GAN: II-A on implementation and II-B on evaluation.
- Section. III: Defense-GAN: III-A on implementation and III-B on evaluation.
- Section. IV: AdvGAN: IV-A on implementation and IV-B on evaluation.
- Section. V: A brief summary of the results and obstacles.

II. APE-GAN

In this section, we briefly discuss the implementation and performance of APE-GAN [1]. The philosophy behind the APE-GAN is to propose an algorithm to eliminate the adversarial perturbation of the input data to defense against the adversarial examples, which can be regarded as a process of learning a mapping from the manifold of adversarial examples to the original image manifold. On the other hand, GAN is designed to generate images similar to the training set and approximate the distribution of data[4]. The authors of the APE-GAN [1] took advantage of the coincidence and similarity between these two problems, proposing to use GAN to do defense.

Emails of the authors: mcding@mit.edu, sirui@mit.edu, dingzw@mit.edu

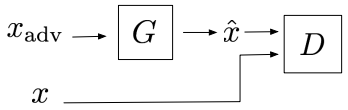


Fig. 1: APE-GAN schematic diagram[1].

A. APE-GAN Implementation

1) Model Architecture and Implementation

The work-flow of the APE-GAN is illustrated in FIG. 1, where G stands for the generator and D represents a discriminator. Here G is trained to alter the perturbation with tiny changes and D is optimized to distinguished the reconstructed images from G from its clean images with the help of a task specified loss function. The implementation is based on the Deep Convolutional Generative Adversarial Nets (DCGAN) [5] owing to its stability.

2) Model Training and Experimental Set-ups

As shown in Fig. 1, both the adversarial and the corresponding clean images are required to train APE-GAN, after which the well-trained G is connected to the front of the classifier to pre-process all the images. The expected result is that the well-trained G can efficiently remove the adversarial perturbation or alter the perturbation to the

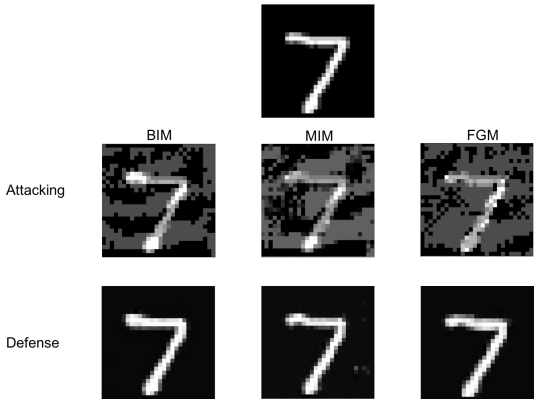


Fig. 2: Adversarial examples of '7' generated by different attacking methods and the corresponding reconstructed images from APE-GAN.

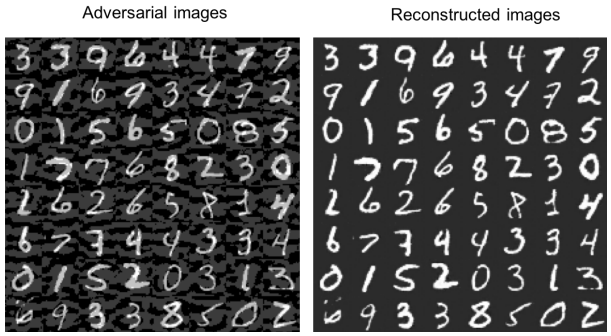


Fig. 3: Adversarial examples generated by different attacking methods and the corresponding reconstructed images from APE-GAN.

TABLE I: Error rates (in %) of three different attacking methods and APE-GAN on MNIST

Attack	Adversarial	APE-GAN
BIM	99.02	3.04
MIM	99.14	3.23
FGM	99.30	2.11

manifold of clean images. In this report, we test the performance of against three different attacking methods implemented in cleverhans [6]. For each attacking method, we used 60000 clean images and their adversarial images as the training set, and another 10000 images are used for testing purpose. It is worth mentioning that no label is required during the training process.

3) Attacking Methods and Visualizations

Due to time limitation, we restrict the study to three attacking methods,i.e., Basic Iterative Method (BIM) [7], Momentum Iterative Method (MIM) [8] and Fast Gradient Method (FGM) [9] and the default hyper-parameters are adopted for all the three methods. As shown in FIG 2, different attacking methods can generate quite different adversarial examples starting from the same clean image, and it turns out that APE-GAN can efficiently eliminate all those perturbation. Here we use '7' as an example, but good performance is observed for images with other labels as shown in Fig. 3.

4) Remarks and Discussion

Based on above visualization figures, the APE-GAN is robust against all the three attack methods. However, no conclusion can be drawn without testing its performance in real classification, which is the focus of the next section. Besides, a recent work [10] reported that the performance of APE-GAN against the CW [11] attack methods is sensitive to the chosen loss function, which is worth investigation in the future. Also, it is necessary to extend our study to a more massive dataset to draw a reliable conclusion.

B. APE-GAN Evaluation

In this section, we discuss the performance of APE-GAN in the aspect of reducing the error rates. At the very beginning, we need a classifier. Unless specified, the classifier is implemented using TensorFlow.

1) Performance against known attacking methods

At the very beginning, we test the APE-GAN against known attacking methods, which means that we train APE-GAN with the adversarial examples generated by the same attacking methods. The performance against different attacking methods are shown in the TABLE I and the error rates are significantly reduced for all the three attacking methods.

2) Performance against unknown attacking methods

In reality, we usually do not know the attacking methods. Thus it is preferred that the defense mechanism could be effective without knowing attacking method. To demonstrate on this point, we test the performance of APE-GAN trained with adversarial examples generated from different attacking methods as shown in TABLE II, where 'All' means that we have used the combined reconstructed images from different methods as the training database. There are a few interesting observations. Firstly, it

TABLE II: Error rates (in %) of APE-GAN trained with different adversarial examples against three different attacking methods on MNIST

Attack/Train	BIM	MIM	FGM	All
BIM	3.04	3.40	2.27	2.53
MIM	6.47	3.23	4.72	3.52
FGM	3.32	4.32	2.11	2.47

TABLE III: Error rates (in %) of APE-GAN trained with different adversarial examples against three different attacking methods on MNIST

Attack/Train	TF	Keras	BCNN
BIM	2.53	2.56	2.54
MIM	3.52	3.48	3.31
FGM	2.47	2.39	2.44

should be noticed that it is not always an advantage to know the attacking method while developing the defense. For example, for the attacking method BIM, the most effective defense, APE-GAN, is trained with adversarial images generated from FGM rather than BIM. Secondly, more extensive training dataset does not guarantee better performance. For all the three attacking methods, the APE-GAN trained with all the adversarial examples never performs best among all the trained APE-GAN.

3) Variation of the methods with different classifiers

As the implementation and training process is independent of the classifier, it is expected that the performance of APE-GAN is independent of the classifier. In this report, we test the performance of APE-GAN with the classifier implemented using TensorFlow, Keras and basic convolutional neural network (BCCN). Error rates are almost unchanged with different implementations of the classifier as shown in TABLE III.

III. Defense-GAN

Defense-GAN [2] works by eliminating the perturbations in adversarial examples and reconstructing the closet clean input in the distribution learned by its generator $G(z)$. When we apply this defense before the classification, the purified input can be classified correctly again, and an input x is then classified as $f(\arg\min_z |G(z) - x|)$ where $f(\cdot)$ is the classifier. Such projection onto the (distribution) manifold learned by $G(z)$ is achieved by a few steps of gradient descent.

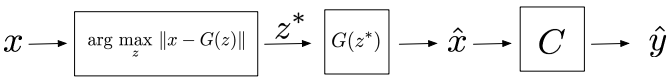


Fig. 4: Defense-GAN schematic diagram [2].

The Defense-GAN is effective in many cases [2], [12] because it follows an Invert and Classify (INC) approach and we cannot back-propagate to train attacks like CW methods.

A. Defense-GAN Implementation

Up to now, the author of [2] haven't released their official implementation. A partial implementation of Defense-GAN [2] is available in the repository of [12], which evaluates its performance against their attacks. However, the code they released is far from a full implementation. First, an essential step of Defense-GAN, the projection onto the manifold of the generator before classification, is missing as an independent module in their released implementation, which forbids us to further evaluate the performance of Defense-GAN against other attacks using the convenient *cleverhans* toolkit. For now, we stick with the attack method in [12]. Although in their articles[12], only WGAN is mentioned. They in fact used pre-trained GANs provided by [13] which includes WGAN-GP[13], WGAN[14], and DC-GAN[15]. Because Defense-GAN is not shown to be effective for CIFAR, the implementation of [12] targets on MNIST.

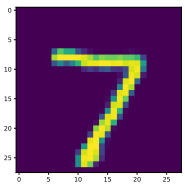
B. Defense-GAN Evaluation

A fatal weakness of Defense-GAN is that: the learned manifold of the generator also contains adversarial examples. We can find them by solving the re-parameterized formulation:

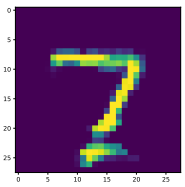
$$\text{min.} \|G(z) - x\|_2^2 + c \cdot l(G(z)) \tag{1}$$

However, such attack based on adversarial example on GAN will not success because the projection is imperfect for the reason due to finite steps of gradient descent. In other words, the projection process may not identify those points.

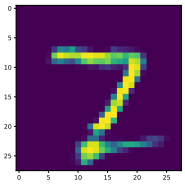
Some defense methods succeed by intentionally obfuscating gradient via approaches like gradient shattering and vanishing/exploding gradients. In [12], a technique called Backward Pass Differentiable Approximation (BPDA) is introduced. [12]



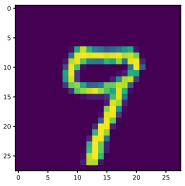
(a) A 7 from test data



(b) An adversarial example misclassified as 2 while using WGAN-GP. The total l_2 distortion is 3.117.



(c) An adversarial example misclassified as 2 while using DCGAN. The total l_2 distortion is 2.673.



(d) An adversarial example misclassified as 9 and while using WGAN. Although the perturbation is large thus it indeed looks like 9. The total l_2 distortion is 5.559.

Fig. 5: Adversarial examples generated by BPDA methods and the corresponding original images from Defense-GAN.

therefore constructs a second attack which utilizes BPDA. Let's regard the output of the purification step in Defense-GAN as a function $g(x)$. $g(x)$ is not neither continuous nor differentiable. However a notable feature of $g(x)$ is that $g(x) \approx x$. The basic principle of BPDA can be understand as approximating ${}_x f(g(x))|_{x=\hat{x}}$ by $\nabla_x | x = g(\hat{x})$. This BPDA approximate provide us with the possibility of gradient calculation and white box attack. Based on it, we can use the standard CW attack.

We generated some adversarial examples using BPDA method, and they are shown in Fig. 5. In those experiments, we use Carlini L_2 attacks with 30000 iterations and a learning rate of 0.1.

1) Defense-GAN Discussion

From our preliminary experimental results, we can see that the perturbations of a considerable amount of adversarial examples generated by methods in [12] are large. This could in principle be a consequence of improper hyper-parameters, but the inaccurate gradient from BPDA approximation is also responsible. The speed of generating adversarial example is relatively slow compared to other methods, while generating one adversarial example for Defense-GAN roughly takes 1 minute. Those features make Defense-GAN an effective defense method. A more detailed evaluation of Defense-GAN utilizing other state-of-art attacks will be carried out once the original authors release their code [1].

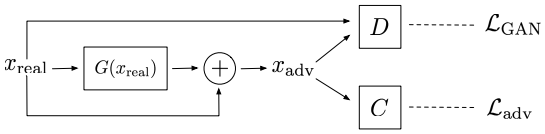


Fig. 6: AdvGAN schematic diagram [3]

IV. AdvGAN

A. AdvGAN Implementation

In this section, we describe our re-implementation of AdvGAN. Since the authors of [3] did not release their code, we re-implement AdvGAN entirely and aim to reproduce some results in the [3].

1) Model Architectures

Generator The generator consists of 4 types of building blocks,

1) Convolution block (Ck):
   ZeroPadding(padding=1) - Convolution(3×3, k filter(s), stride=1) - InstanceNorm - ReLU
2) Downsampling-Convolution block (Dk):
   ZeroPadding(padding=1) - Convolution(3×3, k filter(s), stride=2) - InstanceNorm - ReLU
3) Residual block (Rk):
   ZeroPadding(padding=1) - Convolution(3×3, k filter(s), stride=1) - InstanceNorm - ReLU - ZeroPadding(padding=1) - Convolution(3×3, k filter(s), stride=1) - InstanceNorm - AddingInput
4) Transpose-Convolution block (Tk):
   TransposeConvolution(3×3, k filter(s), stride=2) - InstanceNorm - ReLU

The generator structure consists of C8-D16-D32-R32-R32-R32-R32-T16-T8-T1. It has a strictly symmetric structure, with input and output size 28×28 and bottleneck size 7×7. This auto-encoder like structure helps the generator to map from the benign input to the perturbation to adversary.

[1] According to correspondence with the authors, it should be within a few days.

Discriminator The discriminator has a relatively simple structure, and it has only 2 types of building blocks,

1) Downsampling-Convolution block (Dk): ZeroPadding(padding=1) - Convolution(4×4, k filter(s), stride=2) - InstanceNorm - LeakyReLU(slope=0.2)
2) Fully Connected block (FCn): Flatten - FullyConnected(n layers)

The discriminator structure consists of D8-D18-D32-FC1. The discriminator is much shallower than the generator, as its mission, distinguishing benign and adversarial image, is easier.

Classifier The original paper [3] defines three kinds of classifiers for MNIST. Although their depths are all smaller than 8, they can achieve around 99% accuracy, since MNIST is relatively simple and small. We summarize their structures as following:

1) Classifier A: Convolution(5×5, 64 filters) - ReLU - Convolution(5×5, 64 filters) - ReLU - Dropout(rate=0.25) - Flatten - FullyConnected(128 units) - ReLU - Dropout(rate=0.5) - FullyConnected(10 units) - Softmax
2) Classifier B: Convolution(8×8, 64 filters) - ReLU - Dropout(rate=0.2) - Convolution(6×6, 128 filters) - ReLU - Convolution(5×5, 128 filters) - ReLU - Dropout(rate=0.5) - FullyConnected(10 units) - Softmax
3) Classifier C: Convolution(3×3, 32 filters) - ReLU - Convolution(3×3, 32 filters) - ReLU - MaxPooling(stride=2) - Convolution(3×3, 64 filters) - ReLU - Convolution(3×3, 64 filters) - ReLU - MaxPooling(stride=2) - FullyConnected(200 units) - Relu - FullyConnected(10 units) - Softmax

, where classifier A and B are shallower and Dropouts is used to enables a robust training. Meanwhile, classifier C follows a typical VGG-like architecture and does not use Dropout.

2) Loss Function

Following [3], the complete loss function of AdvGAN is:

$$\mathcal{L} = \mathcal{L}_{\text{GAN}} + \alpha\mathcal{L}_{\text{adv}} + \beta\mathcal{L}_{\text{hinge}} \tag{2}$$

Since it use the least square objective proposed by LSGAN [16], the $\mathcal{L}_{\text{GAN}}$ term should be:

$$\mathcal{L}_{\text{GAN}} = \frac{1}{2}\mathrm{E}_x\left[D^2(x) + (1 - D(x + G(x)))^2\right] \tag{3}$$

For targeted attack, the adversary loss is,

$$\mathcal{L}^{\text{t}}_{\text{adv}} = \mathrm{E}_x\left[\max(\max_{i\neq t} f_i(x) - f_t(x), 0)\right] \tag{4}$$

where $f_i(x)$ is the class score of class i of the classifier, and t is the target class.
For untargeted attack, we propose the corresponding adversary loss as,

$$\mathcal{L}^{\text{u}}_{\text{adv}} = \mathrm{E}_x\left[\max(f_y(x) - \max_{i\neq y} f_i(x), 0)\right] \tag{5}$$

where y is the ground-truth class of input x.
The Soft-hinge loss is to bound the perturbation and stabilize the GAN training. But [3] did not define it clearly. We use,

$$\mathcal{L}_{\text{hinge}} = \mathrm{E}_x\left[\max(||G(x)||_2 - c, 0)\right] \tag{6}$$

and set $c = 0.1$ for MNIST.

3) GAN Training, Hyper-Parameter Tuning, and Experimental Setups

We follow the typical GAN training algorithm. For every iteration, we train the discriminator on 4-8 mini-batches and then train the generator on one mini-batch. When training the discriminator, we do not mix benign and adversary inputs in one mini-batch. Typically, the soft-hinge loss will decrease at first, and then the adversary loss drops. The GAN loss will start to drop after a while when the adversary loss starts to increase again.

Two critical hyper-parameters for training AdvGAN are the two loss weights: α and β. We use grid-search to find a suitable combination. What we find is that $1 \leq \alpha \leq 2$ and $3 \leq \beta \leq 6$ fit all of the three classifiers, under the semi-white box setup.

We use Adam optimizer with a batch size 256 and a learning rate of 0.001 for 1000-2000 iteration. A typical training process of AdvGAN on MNIST takes around 7-10 minutes. Our re-implementation code uses Keras framework and is trained on a Google Cloud Platform virtual machine with 2 Nvidia K80 GPUs.

Fig. 7: Adversarial examples generated by untargeted AdvGAN on MNIST under semi-white box setups against classifier B. The x-axis is classified class, while the y-axis is the ground-truth class.

4) Untargeted Attack

The original paper merely focus on targeted attack. We also implement untargeted attack by changing the adversary loss function $\mathcal{L}_{\text{adv}}^{\text{u}}$. We randomly select the adversarial examples with ground-truth label i but classified to class j for each possible combination of i and j, and visualize them as follows, From Fig. 7, we see that the adversarial samples do not have high perceptual quality as the natural inputs. Although we bound the L_2 error, some adversaries are even difficult to be recognized by humans, which implies that we should re-design our objective $\mathcal{L}_{\text{adv}}^{\text{u}}$, to make AdvGAN works better in this scenario.

5) Targeted Attack

We then proceed to reimplement the targeted attack describe in [3]. After changing back to targeted adversary loss $\mathcal{L}_{\text{adv}}^{\text{t}}$, training of AdvGAN becomes more challenging and more sensitive to the hyper-parameters. However, after tuning, the adversarial examples produced have a higher perceptual quality and relatively smaller L_2 loss. For example, we train a AdvGAN targeting class 0 for classifier C and randomly select adversaries shown in Fig. 8.

6) Average Perturbations

We further compute the average perturbations required for adversaries targeting class j with ground-truth label i, and visualize the distribution as follows, This heat-map can be understood as a fingerprint of the classifier, which reveals the latent weakness of this specific classifier.

Fig. 8: Adversarial examples generated by targeted AdvGAN (target class 0) on MNIST with semi-white box setups against classifier C. The x-axis is classified class, while the y-axis is the ground-truth class.

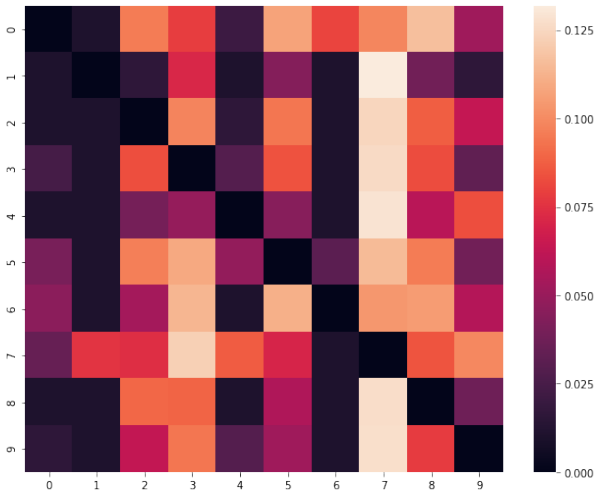


Fig. 9: Average mean-squared loss of perturbations generated by targeted AdvGAN on MNIST under semi-white box setups against classifier C. The x-axis is classified class, while the y-axis is the ground-truth class.

7) AdvGAN Discussion

During our reimplementation process, we re-build the entire code from scratch. We strictly follow the model structures and hyper-parameters listed in the appendices of the original paper [3], if they are clearly defined. However, as some of the configurations are missing, for example, the loss weights α and β and the parameter of soft-hinge loss $\mathcal{L}_{\text{hinge}}$. We cannot guarantee that we correctly re-implement all the details of AdvGAN and can achieve similar performance as stated in the paper [3]. In the

TABLE IV: Attack success rates of adversarial examples generated by targeted AdvGAN (target class 0) on MNIST under semi-white box setups against three classifiers with or without adversarial training.

Model	Defense	Attack Success Rate
A	N/A	93.5%
	Adv. Train	3.2%
B	N/A	92.1%
	Adv. Train	1.1%
C	N/A	98.1%
	Adv. Train	9.3%

proceeding sub-section, we are going to answer this question by evaluating our implementation.

The original paper [3] also describes the method of dynamic distillation for black-box attacks. However, because of time limitation, we will do this in the second-half of the project.

B. AdvGAN Evaluation

In this subsection, we briefly evaluate the performance of AdvGAN by calculating its success rate of attacks on MNIST against three classifiers with or without adversarial training. Due to time limitation, we only perform one trial for each of the experiments. From TABLE. IV we can observe that the two classifiers with dropouts (A and B) are indeed more robust than classifier C. It is consistent with the results in [3]. However, our attack success rate on bare classifier without defense is relatively smaller, especially for classifier B. This may due to that our hyper-parameter is not carefully tuned and AdvGAN does not achieve its best. Overall, our results confirmed the evaluation results in [3].

V. Summary

Finally, we conclude our mid-project report by briefly summarizing the results and obstacles for each of the models so far:

A. Summary of APE-GAN

- APE-GAN results:
  - Partially reproduce the results in the paper [1].
  - Benchmark the performance of APE-GAN against three different attacks implemented in cleverhans [6].
- APE-GAN obstacles:
  - Develop more efficient methods to evaluate the performance of the APE-GAN against different attacking methods even beyond the cleverhans [6].

B. Summary of Defense-GAN

- Defense-GAN results:
  - Dig into the partial implementation of Defense-GAN released by [12].
  - Generate some adversarial examples using CarliniL2 attack with BPDA.
- Defense-GAN obstacles:
  - Current released implementation is too far from complete, which forbids us from further investigation. This problem should be resolved once the original author released their whole code.
  - To attack Defense-GAN efficiently, we will have to customize the calculation of gradients. In another word, toolkit like *cleverhans* is not enough for our need.

C. Summary of AdvGAN

- AdvGAN results:
  - re-implement the model from scratch.
  - train AdvGANs on MNIST to attack three classifiers.
  - achieve un-targeted attack by revising the adversarial loss.
  - evaluate its attack success rates for three classifiers with or without adversarial training.
- AdvGAN obstacles:
  - some unknown hyper-parameters: loss weights α and , constants in adversary loss $\mathcal{L}_{\text{adv}}$ and soft-hinge loss $\mathcal{L}_{\text{hinge}}$.
  - sometimes hard to stabilize the training for targeted semi-white box attack.
  - difficult to train using dynamic distillation for black-box attack.

References

[1] S. Shen, G. Jin, K. Gao, and Y. Zhang, "Ape-gan: Adversarial perturbation elimination with gan," *ICLR Submission, available on OpenReview*, 2017.

[2] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," 2018.

[3] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, "Generating adversarial examples with adversarial networks," *arXiv preprint arXiv:1801.02610*, 2018.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[5] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[6] I. G. R. F. F. F. A. M. K. H. Y.-L. J. A. K. R. S. A. G. Y.-C. L. Nicolas Papernot, Nicholas Carlini, "cleverhans v2.0.0: an adversarial machine learning library," *arXiv preprint arXiv:1610.00768*, 2017.

[7] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[8] Y. Dong, F. Liao, T. Pang, H. Su, X. Hu, J. Li, and J. Zhu, "Boosting adversarial attacks with momentum."

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[10] N. Carlini and D. Wagner, "Magnet and" efficient defenses against adversarial attacks" are not robust to adversarial examples," *arXiv preprint arXiv:1711.08478*, 2017.

[11] ——, "Towards evaluating the robustness of neural networks," in *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 2017, pp. 39–57.

[12] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, 2017, pp. 5769–5779.

[14] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[16] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2813–2821.