Generating Adversarial Examples with Conditional Generative Adversarial Networks

游子慶(Tz-Ching Yu)

指導教授(Advisor)：吳晉賢 教授(Prof. Chin-Hsien Wu)

Department of Electronic and Computer Engineering
National Taiwan University of Science and Technology

1 Introduction

While researchers achieved great success in computer vision tasks using CNNs in the past few years, it has also been pointed out that many machine learning algorithms, e.g. CNNs, fully-connected networks, SVMs, etc., can be easily fooled [3, 17, 19]. By adding carefully constructed perturbations to input samples, one can mislead known models or even unknown models into classifying input samples as erroneous classes. Those carefully designed examples are also known as the adversarial examples. Humans can still correctly classify adversarial examples with their ground truth classes, while those perturbations lead to misclassification for machine learning models. **In this project, we utilize conditional GANs to generate adversarial examples, which is inspired by [7, 20, 21]**, evaluate the generated examples, and compare this method with other commonly used methods, including Fast Gradient Sign Method (FGSM) [3], and Projected Gradient Descent (PGD) [10]. The source code can be found at `https://github.com/kurimulion/Adversarial_examples`.

2 Background

2.1 Neural Networks

Neural networks have been around for a while, though it's only recently that neural networks are widely adopted in the area of machine learning. The most prominent early application of neural networks is the LeNet consisting of only five layers used to recognize handwritten digits. The model achieves high accuracy on MNIST dataset, and it is used to recognize zip codes on letters. Neural networks are function approximators, and they consist of layers of computations. Each layer performs certain operations on the input and passes on the output to another layer. Most commonly, there are convlutional layer, fully-connected layer, and pooling layer. Neural networks with enough layers can be used to approximate any complex functions with some optimization algorithm [5].
In the context of image classification, a neural network takes an input vector, containing raw pixels of an image, and outputs a vector corresponding to scores classes in the classification task, and the prediction of the neural network is calculated as the maximum of the output vector. The formulation is $\hat{y} = \arg\max_i f(x)_i$, where $f(\cdot)$ is the neural network and x is the input sample.

2.2 GANs

Generative Adversarial Networks [2] are a branch of generative models, and GANs have known for producing high-quality and realistic examples compared to other kinds of generative models. The concept of GAN is a two-player game, in which there are discriminator and generator. The discriminator tries to distinguish between real data and fake data generated by the generator, and the generator tries to generate data that fools the discriminator. The two players are competing in a sense, but the discriminator is also teaching the generator how to generate data that mimics data from real distribution. The implementations of generator and discriminator are usually deep neural networks especially deep convolutional neural networks. The generator primarily consists of transpose convolutional layers, and

the discriminator consists of normal convolutional layers. The original loss functions for discriminator and generator are:

$$L_D = -\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))], \tag{1}$$

$$L_G = \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]. \tag{2}$$

While GANs achieve state-of-the-art results in many applications, GANs are also known for their instability of training process. Therefore, besides the original GAN objective function, there are many variants of GAN losses proposed to stabilize the training process. In addition to the slightly modified version [2], there are the hinge version [9] and least squares version [11] of GAN losses.

$$L_D = -\mathbb{E}_{(x,y)\sim p_{data}}[\min(0, -1 + D(x, y))] - \mathbb{E}_{z\sim p_z, y\sim p_{data}}[\min(0, -1 - D(G(z), y))] \tag{3}$$

$$L_G = -\mathbb{E}_{z\sim p_z, y\sim p_{data}} D(G(z), y) \tag{4}$$

2.3 Adversarial Examples

Adversarial examples are described as input images which are applied imperceptible non-random perturbations which change a classifier's predictions [19]. x is a clean example, and p is the perturbation. f is a classifier, and t is a target label. Perturbation is formally defined as:

$$\arg\min_p ||p||_2, \text{subject to } f(x + p) = t \tag{5}$$

However, solving this problem directly is hard, so there have been several methods proposed. Fast Gradient Sign Method (FGSM) [3] does not directly optimize the objective function. FGSM constructs an adversarial example using the sign of the gradient of the target classifier. Since it's possible that examples generated with FGSM are not adversarial examples, meaning that these examples remain correctly classified, there is also an iterative variant of FGSM, I-FGSM, which iteratively perfroms FGSM until adversarial examples are achieved. FGSM works well, if the target model's architecture, weights, and gradient are available. Therefore, it's most suited for white-box attack. Indeed, it could also be used for black-box attack, where there is no information on model's architecture, weight, gradient, etc., with some modifications.

In addition to FGSM, there are also optimization-based methods proposed. Optimization-based methods optimize some objective functions, which indirectly optimize Eq. (5). Among all the objective functions, Eq.(6) shows best result. The formulation is the f_6 proposed in [1] where x' is an adversarial example, and $Z(x) = z$ are the logits which are the output of the classifier before softmax layer. k controls the confidence level of the adversarial example. Example is more likely to transfer to other models with larger k, because the loss is minimized only when the difference of the logit of the target class and the maximum logit among logtis of classes other than the target class is greater than k.

$$\mathcal{F} = \max(\max_{i\neq t}(Z(x')_i) - Z(x')_t, -k) \tag{6}$$

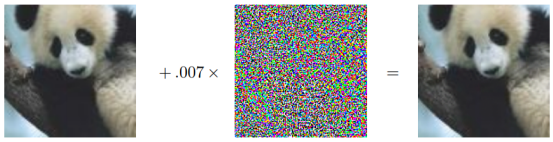


Figure 1: Generation of adversarial examples with FGSM, image captured from [3]

3 Related Work

3.1 AdvGAN

AdvGAN [20] is a Generative Adversarial Network that is trained to generate adversarial perturbation. The idea is to train a GAN to generate perturbation corresponding to a target class given an input image. The AdvGAN adopts architecture similar to pix2pix, and CycleGAN [6, 22] which both take an image and output an image, but the key difference is that the discriminator in AdvGAN takes only the perturbed example or clean example instead of ground truth image and generated image pair, because the ground truth image isn't available in this problem.

4 Method

4.1 GAN Architecture

Other researchers had achieved the task of generating adversarial examples using GANs [7, 20, 21]. We will walk through the differences of each of them, and explain our design choices. CGAN-F and CGAN-adv [21] generate adversarial examples from random noise, while AdvGAN [20] and AdvGAN++ [7] generate only the adversarial perturbations. We think it's more reasonable to generate perturbation for an actual sample from the real data distribution than generating adversarial example from scratch. The major difference between AdvGAN and AdvGAN++ is that AdvGAN++ uses an additional feature extractor to extract the features of an input image and then feeds the feature vector and noise vector to the generator. While AdvGAN++ achieves slightly better result, we may argue that feature extractor seems to be unnecessary and the method may only work well for white-box scenario. Therefore, we adapt the AdvGAN architecture in this project. AdvGAN is a conditional GAN, but, to our knowledge, it's only conditioned on input image instead of both input image and target label. Without target label information, a model can only generate adversarial perturbation for a specific class. In other words, the number of models needed to be trained in order to generate adversarial examples for all classes of a dataset is equal to the number of classes of a dataset. Therefore, we make some modifications to the origianl AdvGAN architecture. We incorporate class information via class conditional batch normalization in generator and projection in discriminator [13]. We also use spectral normalization [12] which normalizes the weights of each layer to stabilize training process. Concretely, the generator

follows U-Net architecture, and the discriminator follows PatchGAN architecture.
We now describe the training process in detail. The generator takes an image from the dataset as input and target class information is incorporated into the process through class conditional batch normalization which has learnable parameters for each class. The target class label is randomly selected which is different from true class label during training. The generator then outputs perturbation corresponding to the mapping between input class and desired target class. We add the perturbation and the input image to obtain the adversarial example. We then apply a per-pixel clipping operation Eq.(7) [8] to the adversarial example to maintain original intensity range and to constrain the L_∞ of perturbation to be at most a preset value ϵ (0.3 in our MNIST experiment and 0.08 in CIFAR10 experiment).

$$Clip(x+p) = \min(1, x+\epsilon, \max(0, x-\epsilon, x+p)) \tag{7}$$

During training, we feed the discriminator {adversarial example, true label} and {clean example, true label} pairs, because we want to encourage the adversarial examples to look like the original class. The class information is taken into account at output layer of the discriminator via projection. We train the GAN with three losses: GAN loss, adversarial loss [1] which is the objective function mentioned earlier, and perturbation loss [20].

$$L_{adv} = \max(\max_{i \neq t}(Z(x+G(x,t))_i) - Z(x+G(x,t))_t, -k) \tag{8}$$

$$L_{per} = \mathbb{E}_{x \sim p_{data}} \max(0, \|G(x,t)\|_2 - \epsilon) \tag{9}$$

GAN loss enforces the adversarial examples to look like data from the dataset. Adversarial loss corresponds to the classification loss of the target class of the target model. Since we are minimizing the loss of the target class for the classifier, the generator learns to generate perturbation such that makes the adversarial example classified as target class. A regularization term is also added to controls the magnitude of the perturbation, ensuring the quality of adversarial examples. We control the importance of each objective through additional hyperparameters, α and β. Thus, the full loss for generator is:

$$L = L_G + \alpha L_{adv} + \beta L_{per} \tag{10}$$

Once the GAN is trained, we can feed samples from the dataset to the generator, and we

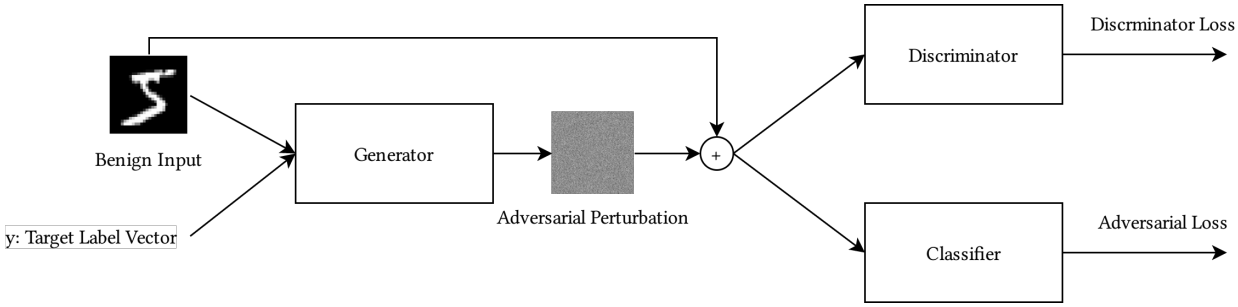


Figure 2: Overall architecture

can then create adversarial examples following the process mentioned above.

5 Experiment

5.1 Black-Box Attack against a MNIST Model

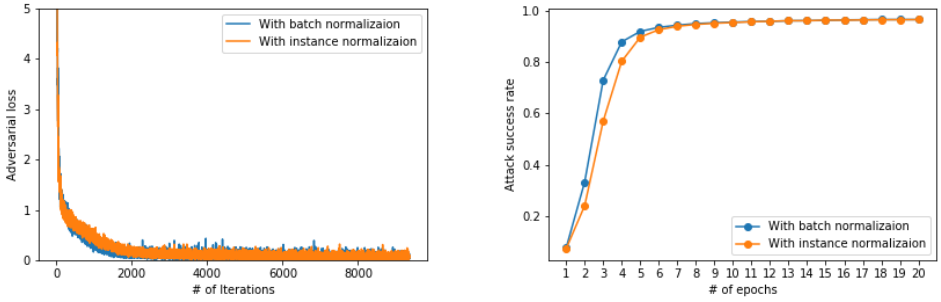
In black-box scenario, an adversary has no access to the target model, including architecture, gradient, parameters, etc. Black-box attack is the most general form of attacks. In real-world situation, the information of models is kept away from users, so it's unlikely that an adversary can have as many details as in the case of white-box scenario. We exploit the transferability of adversarial examples [15, 16]. Therefore, it's likely that adversarial examples we develop using a specific model are also adversarial examples to other models trained to solve the same task.
The MNIST model we use consists of two conv-layers, a fully connected layer, and an output layer which is also a fully connected layer. The architecture is as described in [10].

5.1.1 Result

We use Adam optimizer to train the target model with xavier initialization and train it for 10 epochs over the entire MNIST training set. The model achieve 98% test accuracy. We train the GAN with $\alpha = 10, \beta = 2$ and Adam optimizer with $lr = 0.0004$ for discriminator and $lr = 0.0001$ for generator for 20 epochs. The k in adversarial loss function is set to 0. The generated adversarial examples are shown in Figure 3. We evaluated our method against MNIST Adversarial Examples Challenge in black-box setting. Naturally trained model achieves 85.95% accuracy and adversarially trained model achieves 98.08%.

We thought instance normalization would be a better choice for this task since models with instance normalization achieve better results in style transfer task and image translation task. Therefore, we experiment with batch normalization and instance normalization, but we find model with batch normalization converges faster Figure 4 and produces visually better adversarial examples.

Figure 3: Generated adversarial examples of MNIST
The predictions are: 3, 3, 8, 9, 4, 5, 8, 0, 1, 0, 1, 3, 3, 8, 6, 3.
The target labels are: 3, 3, 8, 9, 4, 5, 8, 0, 1, 0, 1, 3, 3, 8, 6, 3.(from left to right, top to bottom)



(a) Adversarial loss (b) Attack success rate

Figure 4: Models with batch normalization and instance normalization

5.2 Black-Box Attack against a CIFAR10 Model

We also evaluate our method with CIFAR10 Adversarial Examples Challenge. The target model used to classify CIFAR10 dataset is a modified version of ResNet18 [4], and the model achieves 93.57% accuracy on the testing set. The pixels of CIFAR10 images are in the range of $0 - 255$ and the perturbation budget ϵ is set to 8 in the challenge. However, our classification model is trained with pixels in the range of $0 - 1$, so we scale the ϵ to 0.03 to comply with the original budget. The input channels of discriminator is set to 3, and the output of the generator which was originally the output of tanh is multiplied by 0.05 to facilitate the perturbation budget, which significantly increases attack success rate. We again exploit the transferability of adversarial examples to attack the unknown model of the challenge.

5.2.1 Result

We find that while we can achieve high untargeted attack success rate on the testing set, but it's harder to achieve high targeted attack success rate (targeted means the output class equals target class selected by the adversary). After training the GAN for 100 epochs, the targeted attack success rate is around 70% on the target model, while untargeted attack success rate is around 89%. It could be due to the fact the perturbation budget is much smaller ($\epsilon = 0.08$) compared to MNIST Challenge, and samples of CIFAR10 are more complex, so it's harder to achieve high targeted attack success rate. Nonetheless, the generated adversarial examples still perform relatively well. The naturally trained unknown model merely achieves 57.48% accuracy on the generated examples and 86.76% for adversarially trained model. The examples generated are shown in Figure 5.

6 Conclusion

In this work, we develop a way to reliably generate adversarial examples with end-to-end model without iterative optimization. We come to the realization that it's easy to construct adversarial examples, if the model's weight and gradient are exposed to the public. Even if the model details is not available to potential adversary, there are other methods that

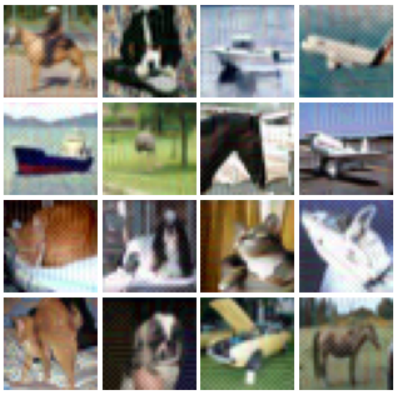


Figure 5: Generated adversarial examples of CIFAR10
The predictions are: 3, 4, 3, 2, 8, 3, 5, 9, 4, 2, 3, 3, 4, 8, 8, 3.
The target labels are: 3, 4, 9, 2, 5, 9, 9, 9, 4, 2, 1, 2, 1, 8, 8, 3.(from left to right, top to bottom)

allow the adversary to generate adversarial examples. There also have been researchers that showed adversarial attacks are possible in the physical environment [8]. Therefore, it's important to be aware of potential adversarial attacks and to introduce countermeasure against such attacks while those applications are being designed.
Potential future work would be to investigate the prevention of adversarial attacks and generation of adversarial examples of larger datasets, such as ILSVRC2012 ImageNet dataset [18].

References

[1] Carlini, N., and Wagner, D. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)* (Los Alamitos, CA, USA, may 2017), IEEE Computer Society, pp. 39–57.

[2] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks.

[3] Goodfellow, I. J., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples.

[4] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

[5] Hornik, K., Stinchcombe, M., and White, H. Multilayer feedforward networks are universal approximators. *Neural Networks 2*, 5 (1989), 359–366.

[6] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5967–5976.

[7] Jandial, S., Mangla, P., Varshney, S., and Balasubramanian, V. Advgan++: Harnessing latent layers for adversary generation. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)* (2019), pp. 2045–2048.

[8] Kurakin, A., Goodfellow, I., and Bengio, S. Adversarial examples in the physical world.

[9] Lim, J. H., and Ye, J. C. Geometric gan.

[10] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks.

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

[12] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks.

[13] Miyato, T., and Koyama, M. cgans with projection discriminator.

[14] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), pp. 2574–2582.

[15] Papernot, N., McDaniel, P., and Goodfellow, I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples.

[16] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Celik, Z. B., and Swami, A. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security* (New York, NY, USA, 2017), ASIA CCS '17, Association for Computing Machinery, p. 506–519.

[17] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)* (2016), pp. 372–387.

[18] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115*, 3 (2015), 211–252.

[19] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014), Y. Bengio and Y. LeCun, Eds.

[20] Xiao, C., Li, B., Zhu, J.-Y., He, W., Liu, M., and Song, D. Generating adversarial examples with adversarial networks, 2019.

[21] Yu, P., Song, K., and Lu, J. Generating adversarial examples with conditional generative adversarial net. In *2018 24th International Conference on Pattern Recognition (ICPR)* (2018), pp. 676–681.

[22] Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2242–2251.