Generating Adversarial Examples with Conditional Generative Adversarial Networks

游子慶(Tz-Ching Yu), 洪昌陞(Hung-Chang Sheng)
指導教授(Advisor)：吳晉賢 教授(Prof. Chin-Hsien Wu)

1 Introduction

While researchers achieved great success in computer vision tasks using CNNs in the past few years, it's a known fact that many machine learning algorithms, e.g. CNNs, fully-connected networks, SVMs, etc., are easily fooled [1, 2, 3]. By adding carefully constructed perturbations to input images, one can mislead known models or even unknown models into classifying input into desired classes. Those carefully designed examples are also known as the adversarial examples. Adversarial perturbations are pretty much imperceptible to human eyes, while those perturbations lead models into misclassification. **In this project, we utilize conditional GANs to generate adversarial examples, which is inspired by [4, 5, 6]**, evaluate the generated examples against some models, and compare this method with other commonly used methods, e.g. Fast Gradient Sign Method (FGSM) [2], and Projected Gradient Descent (PGD) [7]. The source code can be found at https://github.com/kurimulion/Adversarial_examples.

2 Related Work

2.1 Neural Networks

Neural networks have been around for a while, though it's only recently that neural networks are widely used 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 digtis. The model achieves high accuracy on MNIST dataset, and it is used to recognize zip codes on letters. Neural networks are essentially functions, and they consist of layers of computations. Neural networks with enough layers can be used to approximate any complicated functions with some optimization algorithm. In the context of image classification, a neural network takes input image and output probability distribution over a set of classes. The neural network is used to model probability functions, and it's usually done by minimizing a loss function with an optimization algorithm called gradient descend.

2.2 GANs

Generative Adversarial Networks [8] are a branch of generative models, and GANs have known for producing high-quality and realistic examples compared to other kinds of generative models. GANs have been used in many applications, like image generation, super resolution, and style transfer. 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 indistinguishable data. The implementations of generator and discriminator are usually deep neural networks especially deep convolutional neural networks. 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 great 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 loss proposed to stabilize the training process. In addition to the slightly modified version [8], we also experiment with the hinge version [9] and least squares version [10] 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 [1]. 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 we optimize the following objective function Eq.(6) instead. The formulation is the f_6 proposed in [11] 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 clasa is greater than k.

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

There are many methods proposed to generate adversarial examples. Fast Gradient Sign Method (FGSM) [2] constructs an adversarial example using the sign of the gradient of the target classifier. FGSM works, 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.

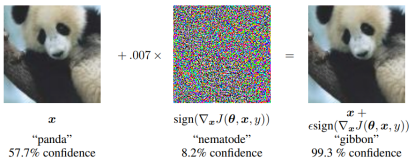


Figure 1: Generation of adversarial examples with FGSM, image captured from [2]

2.4 AdvGAN

AdvGAN [4] 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 [12, 13] 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.

3 Method

3.1 GAN Architecture

[4, 5, 6] all achieve the task of generating adversarial examples using GANs. We will walk through the differences of each of them, and explain our design choices. CGAN-F and CGAN-adv [5] generate adversarial examples from random noise, while AdvGAN [4] and AdvGAN++ [6] 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 [14]. We also use spectral normalization [15] 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 MNIST image 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) [16] 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).

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

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 [11] which is the objective function mentioned earlier, and perturbation loss [4].

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

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

GAN loss enforces the adversarial examples to look like data from MNIST dataset. Adversarial loss corresponds to the classification loss of the target class of the target model. Perturbation loss controls the magnitude of the perturbation. We control the importance of each objective through additional hyperparameters, α and β. Thus, the full loss for generator is:

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

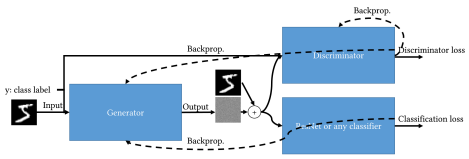


Figure 2: Overall architecture

Once the GAN is trained, we can feed samples from MNIST dataset to the generator, and we can then create adversarial examples following the process mentioned above.

3.2 White-Box Attack against a MNIST Model

In white-box scenario, an adversary has full access to the target model, including architecture, gradient, parameters, etc. We first experiment with white-box attack, because white-box attack is the most basic form of attacks. While in real-world situation it's not exactly possible an adversary would have that much information, we think it's a great way to get started. Also, due to the transferability of adversarial examples [17, 18], 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.

3.2.1 MNIST Model

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 [7]. For more information on detailed architecture, see MNIST_model.py.

3.2.2 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 challenge https://github.com/MadryLab/mnist_challenge in black-box setting, and achieved 85.95% accuracy on naturally trained model and 98.08% on adversarially trained model.

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.

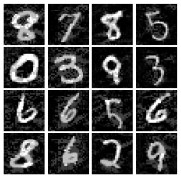
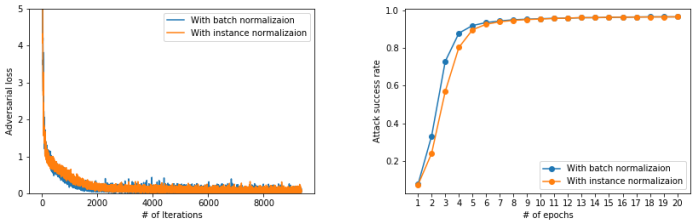


Figure 3: Generated adversarial examples. The predictions are: 3, 3, 8, 9, 4, 5, 8, 0, 1, 0, 1, 3, 3, 8, 6, 3. The target labels are: 3, 3, 8, 9, 4, 5, 8, 0, 1, 0, 1, 3, 3, 8, 6, 3. (from left to right, top to bottom)



(a) Adversarial loss (b) Attack success rate

Figure 4: Models with batch normalization and instance normalization

Conclusion

It's easy to construct adversarial examples if the model's weight and gradient are exposed to the public. Therefore, it's important to be aware of potential adversarial attack while those applications are being designed.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks."

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

[3] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings."

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

[5] P. Yu, K. Song, and J. Lu, "Generating adversarial examples with conditional generative adversarial net."

[6] P. Mangla, S. Jandial, S. Varshney, and V. N. Balasubramanian, "Advgan++ : Harnessing latent layers for adversary generation."

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

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

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

[10] X. Mao, Q. Li, H. Xie, R. Y. K. Lau, Z. Wang, and S. P. Smolley, "Least squares generative adversarial networks."

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks."

[12] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks."

[13] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks."

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

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

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

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

[18] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning."

[19] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks."