智能应用建模　大作业1:对抗样本生成

人工智能学院　薛正海　181220063

1. Background

1.1 What is adversarial example?

Generally, anything that can be fed into a model can be called adversarial examples, as long as they are deliberately revised in order to fool the model. Specifically, the model is usually neural network and examples often pictures with non-trivial noise, since computer vision algorithms with deep neural net are widely researched and heavily employed in practice.

1.2 Why should we study adversarial examples?

Deep neural net is playing such an important role right now that the stakes are very high if it proves inefficient in certain cases. Adversarial examples are carefully designed by researchers to provide such cases where neural net is volunerbly attacked and fooled. Therefore, by studying adversarial examples and coming up with defense approaches, we are fixing an apparent drawback in deep neural net, making it safer to use and less prone to fall short. Also, if we could learn more about the reason why adversarial examples exist, we would take a big step towards understanding deep neural nets, or even proposing better and more robustic models.

2. The attack method

2.1 Motivations and basic idea

We get motivation from the panda-to-gibbon example, where the input image was only slightly modified while the output label transformed completely. Therefore, the basic idea of adversarial attack is to transform the output label with as slight changes to input image as possible. Which of the two requests is more important can either be set previously or get balanced during training.

2.2 Objective function and solution

To put the basic idea formally, let $\mathbf{x}$ and $\mathbf{x}^*$ denote the clean image and the image having being attacked, f denote the mapping exerted by neural net and $\mathbf{t}$ a certain target, we can define the goal of adversarial attack as

$$arg \min_{x^* \in [0,1]} \mathcal{D}(\mathbf{x}^*, \mathbf{x}) - \mathcal{L}(f(\mathbf{x}^*), \mathbf{t}) \tag{1}$$

where $\mathcal{D}$ is a certain function which estimates the distance between inputs and $\mathcal{L}$ a certain loss function. Note the choice of $\mathbf{t}$: if $\mathbf{t} = f(\mathbf{x})$, the attack is in turn untargeted, simply driving the ouput label away from original output $f(\mathbf{x})$. Or $\mathbf{t}$ can be chosen independent of $f(\mathbf{x})$, where a targeted attack takes place, shifting the output as close as possible to target $\mathbf{t}$.

Based on different selection of $\mathcal{D}$ and $\mathcal{L}$, the target (1) has different forms, corresponding to a handful of solutions, respectively. If $\mathcal{D}$ is chosen to be L_∞ norm, the objective function is non-differentiable, and [Goodfellow et al. 2015] transformed it into an optimization problem with constraint, i.e.

$$\arg\max_{\mathbf{x}^* \in [0,1]} \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right), \quad \text{s.t. } \left\|\mathbf{x}^* - \mathbf{x}\right\|_\infty \leq \epsilon \tag{2}$$

which can be solved via Fast Gradient Sign Method(FGSM), also proposed by [Goodfellow et al. 2015]:

$$\mathbf{x}^* \leftarrow \Pi_{[0,1]}\left(\mathbf{x} + \epsilon \cdot \operatorname{sign}\left(\nabla \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right)\right)\right) \tag{3}$$

Note the sign of the gradient and factor ϵ ensure the constraint to be satisfied. Also, [Kurakin et al. 2017] proposed an iterative version of FGSM, iFGSM, which iterates the update several times with smaller timesteps:

$$\mathbf{x}^* \leftarrow \Pi_{\mathbf{x}^* \in \mathcal{S}}\left(\mathbf{x} + \alpha \cdot \operatorname{sign}\left(\nabla \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right)\right)\right) \tag{4}$$

where $\mathcal{S} = \left\{\mathbf{x}^* | [0,1] \cap \left\|\mathbf{x}^* - \mathbf{x}\right\|_\infty \leq \epsilon\right\}$.

[Carlini and Wagner 2017] focues on distance of L_2 norm, forming the loss function as

$$\underset{\mathbf{x}^* \in [0,1]}{\arg\min} \lambda \left\|\mathbf{x}^* - \mathbf{x}\right\|_2^2 - \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right) \tag{5}$$

which is totally differentiable and the update rule reads

$$\mathbf{x}^* \leftarrow \Pi_{[0,1]}\left(\mathbf{x}^* - \alpha\left[2\lambda\left(\mathbf{x}^* - \mathbf{x}\right) - \nabla \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right)\right]\right) \tag{6}$$

In all aforementioned solutions, we expect the neural net f to be well-known and can be differentiated. However, in black-box cases, we can only query the network for the output, while its derivative is totally untractable. Fortunately, we can still approximate the derivative through numerical computations, as proposed by [Chen et al. 2017]:

$$\begin{aligned} \nabla \mathcal{L}\left(f\left(\mathbf{x}^*\right), \mathbf{t}\right) &= \tfrac{\partial \mathcal{L}}{\partial \mathbf{f}} \cdot \tfrac{\partial \mathbf{f}}{\partial \mathbf{x}^*} \\ \left[\tfrac{\partial f}{\partial \mathbf{x}^*}\right]_i &\approx \tfrac{f(\mathbf{x}^* + h\boldsymbol{e}_i) - f(\mathbf{x}^* - h\boldsymbol{e}_i)}{2h} \end{aligned} \tag{7}$$

When implemented, [Chen et al. 2017] utilized stochastic gradient descent and stochasticly chose several coordinates to compute gradient and update.

3. Code setup

3.1 Codes

The code consists of a util file and four algorithm files, where the util file implements the basic training algorithm for mnist and a base class for all adversarial algorithms. The algorithm files implements iFGSM, C&W, targeted C&W and black C&W methods, respectively. All algorithm files can run with python3 *.py except targeted C&W with an optimal command line parameter --target, instructing the target adversarial class. To generate pictures with all desired classes, run `./train.sh`.

3.2 Accuracy on clean examples

Model for mnist dataset stacks convolutional network, max-pooling and batch normalization and uses dropout to regularize. The model was trained 10 batches and achieved training accuracy of 99.34% and testing accuracy 99.29%.

3.3 Generating adversarial images based on each attack method

3.3.1 iFGSM

Only slight changes were made in comparison with the FGSM method: an extra inner loop for iteration is added. [Kurakin et al. 2017] set the hyperparameter $\alpha = 1$, but it turns out in MNIST dataset $\alpha = 0.1$ is enough for adversarial attacks.

3.3.2 C&W

C&W method in the PPT also takes few revisions based on iFGSM, adding to the gradient the derivative of L_2 norm between clean and adversarial pictures. It turned out that α, λ and total gradient steps in (6) exert little influence on the success rate of untargeted attack.

3.3.3 targeted C&W

Targeted C&W algorithm utilizes desired output label and does gradient descent towards it. Note that on average targeted C&W is nine times harder than untargeted attack, therefore more gradient steps or/and greater α is required, and λ should be set to prioritize cost on label mismatch. To accelerate training, an α as big as 10 is set, with λ set to 0.0001. The number of gradient step follows an empirical rule: loop until the cross-entropy loss is less than a certain threshold, then decent adversarial examples should have been generated.

3.3.4 black-box C&W

Black-box C&W method takes advantage of (7) and computes gradient numerically. During training one row of the original picture was chosen as coordinate and did batch gradient descent. The process was extremely slow compared with white-box attack, which makes sense. Note [Chen et al. 2017] chose $h = 0.0001$ in (7), while such a small value caused numerical unstabability during training, so $h = 1$ was set.

3.4 Accuracy of adversarial images and L_2 norm of adversarial noise

	hyperparameter	**accuracy**	**L_2 norm of adversarial noise**
iFGSM	$\epsilon = 2$	80%	$1.77e-7$
	$\epsilon = 4$	82%	$1.17e-6$
	$\epsilon = 8$	86%	$4.88e-6$
	$\epsilon = 16$	90%	$1.98e-5$
C&W	$\alpha = 0.1, \lambda = 1$	92%	$2.16e-10$
	$\alpha = 0.2, \lambda = 1$	92%	$2.87e-10$
	$\alpha = 0.3, \lambda = 1$	100%	$3.12e-10$
targeted C&W	$\alpha = 10, \lambda = 0.001$	100%	0.0019
black-box C&W	$\alpha = 10, \lambda = 1$	90%	$4.70e-5$

Note1: L_2 norm and hyperparameters are based on image pixels $\in [0, 1]$

Note2: Due to memory issues, only 50 test images were extracted to generate adversarial examples.

3.5 Adversarial image presentation

new class	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
original class										

new class	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4										
5										
6										
7										
8										
9										
avg L_2 noise	$3.67e-3$	$1.16e-2$	$1.39e-3$	$9.39e-3$	$1.55e-2$	$1.86e-6$	$9.01e-3$	$4.32e-3$	$7.11e-3$	$3.77e-3$

That's all for my report, thanks for reading!

References

[1] Ian J. Goodfellow, Jonathon Shlens, Christian Szegedy, “Explaining and Harnessing Adversarial Examples”, ICLR 2015 https://arxiv.org/abs/1412.6572

[2] Alexey Kurakin, Ian Goodfellow, Samy Bengio, “Adversarial Examples in the Physical World”, Arxiv 2017 https://arxiv.org/pdf/1607.02533.

[3] Nicholas Carlini, David Wagner, “Towards Evaluating the Robustness of Neural Networks”, Arxiv 2017 https://arxiv.org/pdf/1608.04644.pdf

[4] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, Cho-Jui Hsieh, “ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models”, AISEC 2017 https://arxiv.org/pdf/1708.03999.pdf