Challenges of A.I. : integrity, confidentiality and availability

Practical Course : Crafting adversarial examples and defending against them

1 Introduction

1.1 Goal

In this session of practical course, we aim at developing and analyzing various ways of crafting adversarial examples. The goal of this session is to implement and better understand some concepts that were presented theoretically in the lecture courses. Notably, we will insist on the type of perturbation created by an attack scheme depending on the adversarial capacity and the optimization problem considered in the attack schemes. We will also get more into details regarding the difference between optimization problems for different attacks.

1.2 Prerequisites

For this practical session, you will need: the following Python libraries

- a Deep-Learning Library (Pytorch, Tensorflow, etc.)
- OpenCV
- numpy
- matplotlib
- Scikit-Learn

You can install these libraries with a package manager such as pip or conda.

1.3 Evaluation scheme

The work performed during this session will be graded and will account for 20% of the final grade for the course *Challenges of A.I. : integrity, confidentiality and availability.*
At the end of the session, you will send me a folder containing your work (rbernhard@quantificare.com). Both the code files and the understanding of the concepts presented in the course sessions (questions will be asked during the session) will be taken into account.
During this practical course, I will provide you with some codes files.

2 The FGSM attack

The goal of this part of the session is to implement the FGSM attack [1] for the l_∞ and l_2 norms, and perform some analysis regarding both the norm used as well as the adversary capacity, more precisely the adversary budget.

2.1 Task and data set

We consider a supervised classification task with the MNIST (Modified National Institute of Standards and Technology) data set [2].
This data set is composed of grayscale images of handwritten digits, each image representing a digit from "0" to "9". The task consists in attributing to each input image of a handwritten image the correct digit (label) among the 10 possible. The train set and test set are composed of $60,000$ and

$10,000$ image-label pairs, respectively. Each image is of size 28×28.

Perform an analysis of the data set, notably relatively to class imbalance.

In case of class imbalance, can you image different ways of lowering its importance ?

For the rest of this practical course session, we will not use any of these methods and simply consider the initial training data set as it is.

2.2 Model training

As for the target model, we will consider a simple convolutional model with some final layer (to be determined by you).

1) Train a model with the following architecture:

1. A first 2D convolutional layer with 32 output filters, kernel size $(5, 5)$, stride $(1, 1)$, the *relu* activation function and no bias
2. A max pooling layer with pool size $(2, 2)$ and stride $(2, 2)$
3. A second 2D convolutional layer with 64 output filters, kernel size $(3, 3)$, stride $(1, 1)$, the *relu* activation function and no bias
4. A max pooling layer with pool size $(2, 2)$ and stride $(2, 2)$
5. *What is the appropriate final layer ?*

What is the appropriate loss for such a task ?

If the classification task involved only two labels, what could have been the final layer ? What loss could you have used ?

Split the data set into a training, validation and testing set (initially there is only a train and a test set). Train the model with a batch size of 128, with the Adam optimizer [3] with a learning rate of 0.001.

What is overfitting ? What does indicate overfitting ? Do you know how to reduce overfitting ?

2.3 Model evaluating

Evaluate the model on the MNIST test set.

What is the appropriate metric for such a task ?

2.4 Implementation

Implement the FGSM attack for the l_∞ and l_2 norms.

<u>*Reminder:*</u>
THE FGSM (Fast Gradient Sign Method) is a one-step attack. The considered optimization problem is the following:

$$\begin{aligned} \underset{\alpha}{\text{argmax}} \; & \mathcal{L}(x + \alpha, y, M_\theta) \\ \text{s.t} \quad & \|\alpha\|_p \leq \epsilon \end{aligned}$$

where $\mathcal{L}$ here denotes a classification loss, M_θ the target model and ϵ the adversary budget.

For a clean input-label pair (x, y), the solution is given by:

- for the l_∞ norm:
$$x' = x + \epsilon \operatorname{sign}(\nabla_x \mathcal{L}(x, y, M_\theta))$$
What does this mean in terms of pixel change ? How do pixels change after one FGSM step ?

- for the l_2 norm:
$$x' = x + \epsilon \frac{\nabla_x \mathcal{L}(x, y, M_\theta)}{\|\nabla_x \mathcal{L}(x, y, M_\theta)\|_2}$$

Whatever the norm considered, clipping to the authorized pixel value range $\mathcal{Q}$ is then performed.

Which loss do you think is the most used for $\mathcal{L}$? Can you design another loss ?

2.5 Interpretation

Illustrate the efficiency of the FGSM attack on the MNIST data set.

For both l_∞ and l_2 norms, look at the efficiency difference relatively to the adversary budget.
Link the visual appearance of crafted adversarial examples to remarks you made relatively to the ways pixels are modified during the attack, depending on the norm used.

3 The BIM and PGD attack

The goal of this part of the session is to implement the BIM attack [4] and the PGD attack [5], and look at some difference between these two, notably regarding consequences of the initialization step of the PGD. Moreover, we look into more details the choice of the l_p norm regarding the resulting adversarial perturbation.

3.1 Data set and model

In this section we consider the MNIST data set and the model trained before for this classification task.

3.2 Implementation of the BIM attack

Implement the BIM attack for the l_∞ norm.

Reminder:
THE BIM (Basic Iterative Method) is an iterative attack, consisting in each step at performing the FGSM attack with a certain small step, projecting back on the allowed range of perturbations and pixel value range at each step.
The considered optimization problem is the following:

$$\begin{aligned} &\underset{\alpha}{\operatorname{argmax}}\ \mathcal{L}(x + \alpha, y, M_\theta) \\ &\text{s.t} \quad \|\alpha\|_p \leq \epsilon \end{aligned}$$

where $\mathcal{L}$ here denotes a classification loss, M_θ the target model and ϵ the adversary budget.

For a clean input-label pair (x, y), the solution is given by for the l_∞ norm:

$$\begin{aligned} x^0 &= x \\ \delta^t &= \alpha \operatorname{sign}(\nabla_x \mathcal{L}(x^{t-1}, y, M_\theta)) \\ x^t &= \operatorname{proj}_{B_\infty(x,\epsilon)}(x^{t-1} + \delta^t) \\ x^t &= \operatorname{Clip}(x^t, \mathcal{Q}) \end{aligned}$$

where α is the step size, $\operatorname{proj}_{B_\infty(x,\epsilon)}(x^{t-1} + \delta^t)$ denotes the projection of $x^{t-1} + \delta^t$ onto the l_∞ ball of center x and radius ϵ, and Clip denotes the clipping operation.

3.3 Interpretation

Illustrate the influence on each parameter (ϵ, α, ...) on the final adversarial example, both visually and in terms of misclassification.

3.4 Implementation of the l_∞ PGD attack

Implement the PGD attack in its l_∞ version with the help of the code files I provide you with regarding the initialization step.

3.5 Interpretation

Relatively to the loss you considered for $\mathcal{L}$, notably its final value, illustrate the influence of the existence of the initialization step

3.6 Analysis of the l_1 PGD attack

Reminder:
For the PGD attack in its l_1 version, the solution is given at each step t by:

$$\delta^t = \alpha\, e$$

where e is the unit vector with $e_{i^*} = g_{i^*}$ and $g_{i^*} = \text{argmax}_i \ \text{sign}(\nabla_x \mathcal{L}(x_i^{t-1}, y, M_\theta))$. We then have:

$$x^t = \text{proj}_{B_1(x,\epsilon)}(x^{t-1} + \delta^t)$$
$$x^t = \text{Clip}(x^t, \mathcal{Q})$$

What does this mean in terms of the ways pixels are modified at each step ?

Can you think of another way to modify the l_1 PGD attack in order for it to be faster, even if not optimal ?

4 Transferability

In this section, we illustrate the phenomenon of transferability.

Reminder: The transferability of adversarial perturbations is a powerful tool for an adversary in a black-box setting regarding a target model M_θ. Such adversary trains its own substitute model M_s, and adversarial perturbations crafted on M_s may transfer to the target model M_θ.

4.1 Substitute model training

We consider an adversary having access to the MNIST data set. The target model is the model trained in Section 2.2.

Many factors affect the transferability of adversarial perturbations. What is your intuition about the optimal architecture for the substitute model ? Do you see any other easily tunable factor ?

In accordance with your answers to the previous question, train a substitute model to perform transfer attacks.

4.2 Experiments on transferability

Craft adversarial perturbations on your substitute model with the FGSM, BIM and PGD attacks, for the l_∞, l_1 and l_2 norms.

What do you notice in terms of transferability of the crafted adversarial perturbations ? What seems to be the only viable conclusion on how to tune transferability attacks ?

Can you think of an easy way to modify the BIM attack and enhance adversarial perturbations crafted with it ? Implement it.

5 Adversarial Training

5.1 Implementation

Implement Adversarial Training [5] for the model architecture considered in Section 2.2, for the l_∞ PGD attack.

For the Adversarial Training procedure, you will choose $\epsilon = 0.3$, $k = 40$ and $\alpha = 0.01$.

Reminder:

Algorithm 1 Adversarial Training [5]

Require: adversarial budget ϵ, step size α, number of PGD iterations k, pixel value range $\mathcal{Q}$
Ensure: Model trained with the Adversarial Training [5] defense scheme

Randomly initialize the parameters θ of M_θ
for each epoch: **do**
  for each batch: **do**
    for each input-label pair (x_i, y_i) **do**
      Perform the l_∞ PGD attack for k iterations:
      $x_i^0 = x_i + \eta \quad$ with $\eta \sim \mathcal{U}(-\epsilon, \epsilon)$
      for $t = 1 \dots k$ **do**
        $\delta^t = \alpha \operatorname{sign}(\nabla_{x_i} \mathcal{L}(x_i^{t-1}, y, M_\theta))$
        $x_i^t = \operatorname{proj}_{B_p(x_i,\epsilon)}(x_i^{t-1} + \delta^t)$
        $x_i^t = \operatorname{Clip}(x_i^t, \mathcal{Q})$
      end for
      $x_i' = x_i^k$
    end for
  end for
  Calculate gradients and update the loss function:
  $\theta \leftarrow \theta - \eta \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \mathcal{L}(x_i', y_i, M_\theta)$
end for
return P

5.2 Evaluation

Illustrate the robustness of your model trained with Adversarial Training against different types of attacks.

References

[1] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[4] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *International Conference on Learning Representations*, 2016.

[5] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *International Conference on Learning Representations*, 2018.