Exercise 02

Adversarial Examples, Defenses and Box Verification

Reliable and Trustworthy Artificial Intelligence
ETH Zurich

Problem 1 (Projection onto ℓ_p-balls)**.** The Euclidean projection $\boldsymbol{z} \in \mathbb{R}^n$ of a point $\boldsymbol{y} \in \mathbb{R}^n$ onto the ϵ-sized ℓ_p-ball around $\boldsymbol{x} \in \mathbb{R}^n$ is defined as (note the ℓ_2 and ℓ_p norms):

$$\boldsymbol{z} = \underset{\boldsymbol{x}' \text{ s.t. } \|\boldsymbol{x}'-\boldsymbol{x}\|_p \leq \epsilon}{\arg\min} \|\boldsymbol{x}' - \boldsymbol{y}\|_2 \tag{1}$$

In general, this is a hard problem and exact closed-form solutions are only known for few p. In this task, we investigate this problem for different p.

a) Consider $p = \infty$. Derive a closed-form formula for projection onto the ϵ-sized ℓ_∞-ball around $\boldsymbol{x} \in \mathbb{R}^n$.

b) Consider $p = 2$.

   (i) Derive a closed-form formula for the projection $\boldsymbol{z} \in \mathbb{R}^n$ of a point $\boldsymbol{y} \in \mathbb{R}^n$ onto the ϵ-sized ℓ_2-ball around $\boldsymbol{x} \in \mathbb{R}^n$.

   (ii) Prove that for $n = 2$, your closed-form solution is correct, i.e., show that there exists no $\boldsymbol{q} \neq \boldsymbol{z}$ in the ϵ-sized ℓ_2-ball around $\boldsymbol{x}$ that is closer to $\boldsymbol{y}$ than $\boldsymbol{z}$.

   Hint: Assume for the sake of contradiction that there exists such a point $\boldsymbol{q}$. Use the triangle inequality.

c) Consider a general $p \geq 1$. Instead of finding an exact solution, we are only looking for an *approximate* projection.

   (i) Assume $\|\boldsymbol{x} - \boldsymbol{y}\|_p > \epsilon$. Like for $p = 2$, we can try to move $\boldsymbol{y}$ closer to $\boldsymbol{x}$ along a direct line by shifting the former along the vector $\boldsymbol{y} - \boldsymbol{x}$ rescaled according to p. Use this idea to derive a closed-form formula for approximate projection, where the result $\boldsymbol{z}$ is guaranteed to lie in the ϵ-sized ℓ_p-ball around $\boldsymbol{x}$ (but is not necessarily the closest point).

(ii) Construct a concrete counter-example for $n = 2$ showing that for $p = 1$, your formula is only approximate and does not solve (1).

Note: For general p, we can alternatively try to solve (1) using expensive iterative gradient-based optimization algorithms.

Problem 2. In the lecture, we discussed the Carlini-Wagner optimization problem [1]:

$$\begin{aligned}&\text{find } \boldsymbol{\eta}\\&\text{minimize } \|\boldsymbol{\eta}\|_p + c \cdot \operatorname{obj}(\boldsymbol{x} + \boldsymbol{\eta})\\&\text{such that } \boldsymbol{x} + \boldsymbol{\eta} \in [0,1]^n\end{aligned} \tag{2}$$

Directly optimizing this objective with gradient-based methods can be problematic in the case of $p = \infty$. In this task, we will investigate this and discuss a surrogate term for $\|\cdot\|_\infty$. In the following, assume that $\boldsymbol{x}, \boldsymbol{\eta} \in \mathbb{R}^n$. We define $h(\boldsymbol{\eta}) = \|\boldsymbol{\eta}\|_\infty$ and $g_\tau(\boldsymbol{\eta}) := \sum_{i=0}^{n-1} \max(\boldsymbol{\eta}_i - \tau, 0)$ for $\tau \in \mathbb{R}$.

a) Calculate the partial derivatives $\frac{\partial}{\partial \boldsymbol{\eta}_i} h(\boldsymbol{\eta})$ and $\frac{\partial}{\partial \boldsymbol{\eta}_i} g_\tau(\boldsymbol{\eta})$.

b) Instantiate the above derivatives with $\tau = 0.9$ and $\tau = 2.0$ for

$$\boldsymbol{\eta} = (1.00005, 1.00004, 1.00003, 1.00002, 0.001, 0.001)^\top.$$

c) Assume we are minimizing $h(\boldsymbol{\eta})$ using gradient descent with step size $\gamma = 0.01$. How many iterations are required until $h(\boldsymbol{\eta}) < 0.01$? What is the problem?

d) When combining $h(\boldsymbol{\eta})$ with the objective function to obtain the full optimization problem in (2), optimizing the latter using gradient descent may actually result in an infinite number of iterations. Explain why this can happen.

e) To circumvent the above problems, we can use g_τ as a surrogate for h. How does g_τ improve the situation?

Note: For the reasons discussed above, the authors of [1] suggest to use g_τ instead of h, where τ is lowered repeatedly during optimization whenever $\max_i(\boldsymbol{\eta}_i) \leq \tau$ in order to produce a solution $\boldsymbol{\eta}$ with small ℓ_∞-norm.

Problem 3 (Coding)**.** In the ZIP file provided on the course webpage, you can find a python skeleton `task3.ipynb` along with a pre-trained MNIST classifier `model`.

Note: The skeleton is based on the PyTorch[1] framework. We strongly recommend that you familiarize yourself with PyTorch now, because the course project will rely heavily on PyTorch. This exercise allows you to gain some initial experience with PyTorch.

Re-using your code from last week, implement the (untargeted) PGD attack ([2]) discussed in the lecture (`pgd`). Here, `k` is the number of FGSM iterations with `eps_step` step size. Each FGSM iteration is projected back to the `eps`-sized ℓ_∞-ball around `x`. Your implementation should clamp the adversarial example back to the image domain.

Problem 4 (Coding)**.** In this task, you are going to implement adversarial training with PGD (originally introduced in [2]) and an alternative defense.

a) Complete the provided code skeleton in `task4.py` to train the network `model` with PGD defense. That is, for data distribution D (the MNIST dataset in our case) and network parameters θ, optimize the following objective:

$$\min_\theta \; \mathbb{E}_{(x,y)\sim D}\left[\max_{x'\in\mathbb{B}_\epsilon(x)} L(\theta, x', y)\right]. \tag{3}$$

Here, $\mathbb{B}_\epsilon(x) := \{x' \mid \|x - x'\|_\infty \leq \epsilon\}$ denotes the ϵ-sized ℓ_∞-ball around x. L is the usual classification loss $L(\theta, x', y) := \mathcal{H}(y, f_\theta(x'))$, where $f_\theta = (\text{softmax} \circ \texttt{model})$ denotes the output distribution of the neural network, and $\mathcal{H}$ the cross entropy[2] between distributions (being a discrete value, we treat y as a one-hot distribution). In PyTorch, you can use `nn.CrossEntropyLoss`[3] to implement L.

Use PGD to solve the inner optimization problem with $\epsilon = 0.1$, $k = 7$ steps, and $\epsilon_{\text{step}} = 2.5\frac{\epsilon}{k}$. You can reuse your implementation of untargeted PGD from the previous exercise, or create a more efficient (batched) version better suited for training.

Compare the accuracy results with and without PGD training.

b) The TRADES [3] algorithm minimizes the following objective (see [3] for details):

$$\min_\theta \; \mathbb{E}_{(x,y)\sim D}\big[\underbrace{L(\theta, x, y)}_{\text{for accuracy}} + \underbrace{\lambda \max_{x'\in\mathbb{B}_\epsilon(x)} L(\theta, x', f_\theta(x))}_{\text{regularization for robustness}}\big]. \tag{4}$$

[1] https://pytorch.org

[2] https://en.wikipedia.org/wiki/Cross_entropy

[3] https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

Extend your implementation in `train.py` to the TRADES defense. Again, the inner optimization problem is solved using PGD. Use $\lambda = 1.0$ and the same parameters as in the previous task. Compare your results with the previous task.

Note: Here, $L(\theta, x', f_\theta(x))$ still denotes the cross entropy loss.

References

[1] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 39–57. DOI: 10.1109/SP.2017.49. URL: https://arxiv.org/abs/1608.04644.

[2] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[3] Hongyang Zhang et al. "Theoretically Principled Trade-off between Robustness and Accuracy". In: *ICML*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7472–7482.