Exercise 03

Adversarial Examples

Reliable and Interpretable Artificial Intelligence
ETH Zurich

Problem 1 (Coding)**.** This task can either be done either in `task.py` or `task.ipynb` (whichever you prefer). You are provided access to an MNIST classifier `model`, which is the ConvNet introduced in the previous excercise session in `mnist_train.ipynb` trained for 50 epochs (accuracy of 0.9934). Instead of this provided model you can also use your own MNIST classifier.

1. Implement targeted FGSM. The signature of your implementation should be similar to `fgsm_targeted(model, x, target, eps)`. Your implementation should also clamp back to the image domain (i.e. $[0, 1]^{28\times 28}$ for MNIST). Feel free to extend the signature as needed.

2. Implement untargeted FGSM. Similar to the targeted attack, but instead of a `target` the correct `label` is passed.

3. Implement iterated, projected FGSM (also known as PGD attack) in both targeted and untargeted setting. This algorithm performs k iterations of FGSM with perturbation magnitude ϵ_s each. After each step the current solution is projected back to the ϵ sized L_∞-ball around the initial starting input. Note that projection to the L_∞-ball can be obtained by just clipping values. Use your solutions from the previous two tasks. The signature should look like `pgd_targeted(model, x, target, k, eps, eps_step)` where `eps` is the size of the ϵ-Ball to be projected on and `eps_step` is the size for an individual FGSM step. Again you should clip to the image domain. The signature for the untargeted case has the same arguments.

4. **Optional:** FGSM and PGD attacked images for MNIST do not look very impressive as the perturbation is clearly visible. Run your attacks also for CIFAR-10. If you implemented your attacks correctly you should not need to change anything but datapoint `x` and the `model` passed to the function. You will see that there the attacks can be hidden much more in the image.

Problem 2. [1] state the following (notation adapted to lecture notation):

We define an objective function obj *such that* $f(\boldsymbol{x}+\boldsymbol{\eta}) = t$ *if and only if* $\text{obj}(\boldsymbol{x}+\boldsymbol{\eta}) \leq 0$. *There are many possible choices for* obj*:*

$$\text{obj}_1(x') = -\,\text{loss}_t(\boldsymbol{x}') + 1$$
$$\vdots$$

Note that $f(x) = \text{argmax}_k\, N(\boldsymbol{x}')_k$ denotes a neural network making a classification. $N(\boldsymbol{x})$ is the actual computation of the network (including a softmax layer) and $f(\boldsymbol{x})$ is syntactic sugar for also making the classification. Consider $\text{loss}_t(\boldsymbol{x}')$ to be the cross-entropy loss on the neural network output $N(\boldsymbol{x}')$, with t as the target label. Since information is given in bits, the logarithm with base 2 is commonly used for cross-entropy.

1. Show that theorem is wrong by giving a counterexample.
2. To correct this oversight you will need to adjust the definition of $\text{obj}(\boldsymbol{x}')$ and drop one of the directions of the if-and-only-if in the theorem. Make these adjustments and state the new function along with the correct theorem.

Problem 3. In the lecture we also looked at the optimization problem phrases by [1]:

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

Optimizing the norm directly can be problematic, especially in the case of $\|\cdot\|_\infty$. In this task we will investigate this and a surrogate term. To simplicity the notation we will assume that $\boldsymbol{x}$ and $\boldsymbol{\eta}$ are n-vectors here (although they are usually matrices representing images). We define $h(\boldsymbol{\eta}) = \|\boldsymbol{\eta}\|_\infty$ and $g(\boldsymbol{\eta}) = \sum_{i=0}^{n} \max(\boldsymbol{\eta}_i - \tau, 0)$ for some constant τ.

1. Calculate $\frac{\partial}{\partial \boldsymbol{\eta}} h(\boldsymbol{\eta})$.
2. Calculate $\frac{\partial}{\partial \boldsymbol{\eta}} g(\boldsymbol{\eta})$.

3. Instantiate the above derivatives for $\boldsymbol{\eta} = (1.00001, 1.0, 1.0, 1.0, 0.001, 0.001)^T$ and for $\tau = 0.9$ and $\tau = 2.0$.

4. What is a problem when minimizing $h(\boldsymbol{\eta})$ with gradient decent?

5. Does $g(\boldsymbol{\eta})$ suffer the same problem?

References

[1] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *arXiv preprint arXiv:1608.04644* (2016).