# Security and Privacy in Machine Learning

Homework 4

Javad Hezareh
98101074

Spring 2023

## Contents

## 1 Universal Adversarial Perturbations

**(a)** Adversarial perturbation found by the UAP is universal in the sense that using this single perturbation one can fool the model to categorize images in our dataset as false. In other words, UAP's output generalizes over our dataset.

**(b)** Finding a universal attack for a model is important cause it will help to train a robust model and improve decision boundary by using only a single example instead of crafting an example for each input.

**(c)** To find a universal perturbation we want to almost fool every data point by adding a single perturbation. Also, we have an upper bound on the norm of our perturbation. Therefore we can model this problem as

$$\begin{aligned} \arg\max_{\boldsymbol{v}} \quad & \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}[g(\boldsymbol{x}+\boldsymbol{v})] \\ \text{s.t.} \quad & \|\boldsymbol{v}\|_p \leq \epsilon \end{aligned}$$

**(d)** Practical (submitted with HW5)

## 2 Physical Adversarial Attacks

**(a)** The goal of this paper is to craft adversarial patches in the real physical world to fool a person detector system. To do so they build a small patch about 40cm×40cm which can be carried by a person who wants to hide himself from a surveillance camera and the system will not detect him as a person.

**(b)** Crafting adversarial examples in the physical world has some limitations. At first, it might seem easy to use the digital examples in the real world but in practice, you can find successful digital adversarial examples not so robust in the physical world. One of these examples is the limitation on printing images. Printers can not print all real-valued pixels we have in digital systems therefore the printed image might not be successful to fool the system. To solve this problem, one can use loss functions consider this problem, such as non-printability score (NPS) loss[1], in the process of building digital adversarial examples.

**(c)** A digital adversarial example might be successful to fool a classification model in an especial configuration such as viewpoint or brightness. Using a patch of this image in the physical world to attack a model that continuously observes the environment is not effective because we need to build exactly that configuration again. To solve this problem we can use the Expectation of Transformation (EoT) method to find a physical adversarial example robust to different transformations.

**(d)** In this method we want to maximize the expectation of the log-likelihood given transformed inputs and yet the expectation of distance between generated example and input be bounded. Therefore we can formulate this problem as

$$\begin{aligned} \arg\max_{\boldsymbol{x}} \quad & \mathbb{E}_{t\sim\mathcal{T}}[\log(\mathbb{P}(y_t|t(\boldsymbol{x})))] \\ \text{s.t.} \quad & \mathbb{E}_{t\sim\mathcal{T}}[d(t(\boldsymbol{x}), t(\boldsymbol{x_0}))] \leq \epsilon \end{aligned}$$

[1] Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition

## 3 Robustness Evaluation of Defenses

**(a)** One of the interesting results of this evaluation that triggers doubts about the performance of the defense is the superior performance of the black-box attack rather than the other white-box attacks. This result raises the question that whether this defense truly enhances the robustness of the model or just uses gradient obfuscation to prevent users from crafting adversarial perturbations. Therefore we need to do other evaluations on this method before accepting it as a successful defense. To do so, one can examine the unbounded attacks to test if the model accuracy decreases to 0% or not. If the accuracy doesn't drop to 0% by using unbounded attacks, it is a sign that model just uses gradient obfuscation rather than boosting inherent robustness. For further evaluation, one can perform a random attack to find adversarial perturbations. If the accuracy of the model decreases more than other evaluated attacks, it is also another sign of gradient obfuscation.

**(b)** In this paper the authors have proposed two new attacks based on PGD and then combined these two with another two methods to build the *AutoAttack*.

- **Auto-PGD**: The classical PGD algorithm uses fixed step size and hence one needs to tune this parameter for each new defense. Auto-PGD (APGD) method updates its step size parameter based on the progress of the optimization and the overall budget. To do so they build some checkpoints iteration to update step size. In each checkpoint, the algorithm checks the progress of optimization and the overall budget and decides whether it is necessary to halve the current step size. It starts with $\eta_0 = 2\epsilon$. The other difference with classic PGD is in the update step where APGD uses momentum.
- **PGD with an alternative loss**: The classical PGD algorithm uses the Cross-Entropy loss function as the objective function. This function is invariant to shifts of the logits but not to rescaling. Its gradient w.r.t input has the same properties. Not being invariant to rescaling makes gradient vanishing possible and hence the algorithm might not find a good adversarial point. To solve this problem, the authors propose a new loss function called Difference of Logits Ratio (DLR) which is invariant to both shifts and rescaling.
- **AutoAttack**: This attack is the main contribution of the paper that is a combination of the previous methods with two other attacks, FAB and Square Attack. They called it AutoAttack since it doesn't need any parameter tuning. They use the following attacks to compose AutoAttack: (1) $\text{APGD}_{\text{CE}}$ which is APGD with CE loss function, (2) $\text{APGD}^{\text{T}}_{\text{DLR}}$ which is APGD with Targeted-DLR loss function, (3) targeted version of FAB and (4) Square Attack with one run of 500 queries. FAB is a white-box attack that tries to minimize the norm of the perturbation and relies on the gradient of the logits. Square Attack is a score-based black-box attack that uses random search to find adversarial examples.

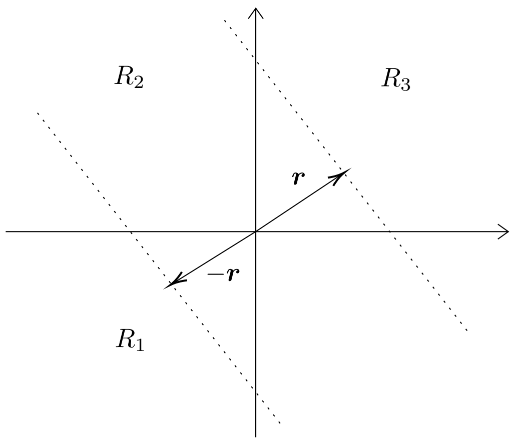
## 4 Certified Robustness

**(a)** Using $f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x} + b)$ as our model means the decision boundary is a surface in the input space. Therefore for an input $\boldsymbol{x}$ the certified robustness radius will be the distance of this point from the decision surface. Any point in this circle around a given point will still be on the same side as the input point and hence the model's output doesn't change. To find the distance of $\boldsymbol{x}$ from the decision surface we can use $\boldsymbol{x} = \boldsymbol{x_s} + \boldsymbol{r}$ where $\boldsymbol{x_s}$ is the projection of $\boldsymbol{x}$ on the surface and $\boldsymbol{r}$ is a vector perpendicular to the surface. Therefore:

$$\begin{aligned}
& \boldsymbol{x} = \boldsymbol{x_s} + \boldsymbol{r} & \\
\implies\ & \boldsymbol{w}^T\boldsymbol{x} + b = \boldsymbol{w}^T\boldsymbol{x_s} + \boldsymbol{w}^T\boldsymbol{r} + b & (\boldsymbol{w}^T(\cdot) + b) \\
\implies\ & f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{r} & (\boldsymbol{x_s} \text{ is on the surface}) \\
\implies\ & f(\boldsymbol{x}) = \|\boldsymbol{w}\|_2\|\boldsymbol{r}\|_2 & (\boldsymbol{r} \text{ is perpendicular to the surface}) \\
\implies\ & \|\boldsymbol{r}\|_2 = \frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|_2} &
\end{aligned}$$

Therefore the certified robustness radius for an input $\boldsymbol{x}$ is $\frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|_2}$.

**(b)** Intuitively for a given point $\boldsymbol{x}$ which has been classified as $c$ by classifier $f$ and if $\boldsymbol{r}$ be the perpendicular vector from decision surface to $\boldsymbol{x}$ we can divide the space of possible outcomes of $\mathcal{N}(\boldsymbol{0}, \sigma^2\boldsymbol{I})$ to three regions.



For a noisy input if $\boldsymbol{\epsilon}$ lies in the $R_1$ region then $f$ prediction on $\boldsymbol{x} + \boldsymbol{\epsilon}$ will change. Therefore for $\boldsymbol{x}$ and its predicted class $c$ by $f$ we have:

$$\begin{cases}
\mathbb{P}(f(\boldsymbol{x}+\boldsymbol{\epsilon}) = c) = \mathbb{P}(\boldsymbol{\epsilon} \in R_2) + \mathbb{P}(\boldsymbol{\epsilon} \in R_3) \\
\mathbb{P}(f(\boldsymbol{x}+\boldsymbol{\epsilon}) = \neg c) = \mathbb{P}(\boldsymbol{\epsilon} \in R_1)
\end{cases}$$

As we use symmetrical Gaussian noise then we have $\mathbb{P}(\boldsymbol{\epsilon} \in R_3) = \mathbb{P}(\boldsymbol{\epsilon} \in R_1)$, therefore:

$$\mathbb{P}(f(\boldsymbol{x}+\boldsymbol{\epsilon}) = c) = \mathbb{P}(\boldsymbol{\epsilon} \in R_2) + \mathbb{P}(\boldsymbol{\epsilon} \in R_1) \geq \mathbb{P}(\boldsymbol{\epsilon} \in R_1) = \mathbb{P}(f(\boldsymbol{x}+\boldsymbol{\epsilon}) = \neg c)$$

This means by adding symmetrical Gaussian noise to the inputs the probability of change in the prediction is always less than or equal to the probability of not change in the prediction. Therefore $g$ will always predict the same class as $f$ predicts.