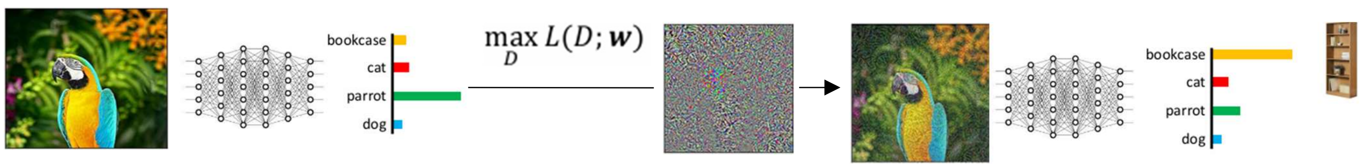
PART 1

Adversarial attack

Adversarial attacks exploit the same underlying mechanism of learning, but aim to maximize the probability of error on the input data: $\max_D L(D; \boldsymbol{w})$
I add a perturbation **ε** that is an adversarial noise for "evading" the ML system. I compute **ε** with the **gradient descent algorithm** which is then added to the input image to cause misclassification



Pattern recognition as "classification"

Pattern classification is about assigning class *labels* to patterns. Patterns are described by a set of measurements called also **features** (or attributes, inputs). If we are working with image data, **feature values** could correspond simply to the **brightness** of each **pixel**.

pattern
Pattern Classification
Class= seven

We assume that each pattern is described by a *feature vector* with "**d**" elements: **x = (x1,x2,....,xd).**

Class: intuitively, a class contains similar patterns, whereas patterns from different classes are dissimilar (e.g., dogs and cars).

We denote them as: **Ω = {ω1, ω2, ..., ωc}**, each pattern belongs to one of the "c" classes of the set Ω. We say that each pattern has a class **label**.

Hand-crafted vs. non-handcrafted (learned) features

The Handcrafted features are manually engineered by the human designer. Non-handcrafted features are automatically learned from a ML algorithm.

BAYES THEOREM

$\hat{P}(\omega_i/\bar{x}) \longleftarrow$ COMPUTING THIS PROBABILITY IS (OFTEN) QUITE/VERY DIFFICULT

$\boxed{P(A/B) = P(B/A) - P(A)}$

LIKELIHOOD

$$\boxed{P(\omega_i/\bar{x}) = \frac{P(\bar{x}/\omega_i)\cdot P(\omega_i)}{\cancel{P(\bar{x})}}}$$

CLASS PRIOR PROBABILITY

$\boxed{P(x) = \sum_i P(x|t_i)\, P(t_i)}$

EVIDENCE

↳ P(x) IS NOT NECESSARY PROBABILITY

In general, a classification model can be regarded as a function $\boldsymbol{f(x)}$, that takes as input the vector x (representing the pattern) and provides as output the classification (class label).

The classification model could be a linear function $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b = \sum_{j=1}^{d} w_j x_j + b$

We use **loss functions** to evaluate how good (or bad) our classification models are, learning the "optimal" values of ***w,b*** in order to minimize the **error probability**.

D: training set containing «n» examples
y_i is the class label for training example $\boldsymbol{xi}$
$f(x_i, \boldsymbol{\theta})$: is the classification model

$$L = (D, \boldsymbol{\theta}) \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

Optimization algorithms

The most popular optimization algorithms follow an approach called **gradient descent**. We need to find the best possible parameters for minimizing the *loss function*.

$$\boldsymbol{w}^\star, b^\star = \underset{\boldsymbol{w},b}{\operatorname{argmin}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i))}_{\substack{\text{loss term} \\ L(D,\boldsymbol{\theta})}} + \lambda \underbrace{\Omega(\boldsymbol{w})}_{\substack{\text{regularization term} \\ \Omega(\boldsymbol{w})}} \xrightarrow{} \text{AVOID OVERFITTING}$$

λ: regularization hyperparameter

Likelihood Ratio Test (MAP decision rule written in another way)

$$P(w_1/\bar{x}) \gtrless P(w_2/\bar{x})$$

$$\frac{P(\bar{x}/w_1) \cdot P(w_1)}{\cancel{P(\bar{x})}} \gtrless \frac{P(x/w_2) \cdot P(w_2)}{\cancel{P(\bar{x})}} \implies \frac{P(\bar{x}/w_1)}{P(\bar{x}/w_2)} \gtrless \frac{P(w_2)}{P(w_1)}$$

Example of open question: Let us suppose that we want to discriminate between normal and intrusive network traffic, namely, **two** data classes ω_N, normal traffic, and ω_{INT}, intrusive network traffic. We suppose to use a single feature **x** to characterize traffic data (one-dimensional feature space), and we assume that the model of the network traffic is the following:

$$P(\omega_N)=\frac{1}{2};P(\omega_{INTR})=\frac{1}{2}$$

$$p(x/\omega_i)=\frac{1}{\sqrt{2\pi}\,\sigma}\exp\left[-\frac{1}{2}\left(\frac{x-\mu_i}{\sigma}\right)^2\right]; \nearrow \text{GAUSSIAN DISTRIBUTION}$$

$$\mu_N=0;\mu_{INTR}=4;\quad \sigma_N=\sigma_{INTR}=1;$$

<u>Question</u>: **Find the decision regions using the likelihood ratio test and compute the total error probability.**

$$l(x)=\frac{p(x/\omega_N)}{p(x/\omega_{INTR})}\ \underset{\omega_{INTR}}{\overset{\omega_N}{\substack{>\\<}}}\ \frac{P(\omega_{INTR})}{P(\omega_N)}=\theta$$

$$\theta=\frac{P(\omega_{INTR})}{P(\omega_N)}=1$$

$$l(x)=exp\left[\frac{1}{2}\left(\left(\frac{x-4}{1}\right)^2-\left(\frac{x-0}{1}\right)^2\right)\right]=$$

$$exp\left[\frac{1}{2}(x^2+16-8x-x^2)\right]=$$

$$=exp[8-4x]$$

$$l(x)=\theta\Longrightarrow exp[8-4x]=1\Longrightarrow$$

$$8-4x=\ln(1)\Longrightarrow \underline{x_b=2}$$

Total error probability:

Two components of error probability: (intrusions wrongly labeled as normal traffic) + (normal traffic wrongly labeled as intrusion) *(the actual value of the integrals can be found by looking at the table of values of the <u>erf function</u>)*

$$P\{x\in R_N,x\in\omega_{INTR}\}+P\{x\in R_{INTR},x\in\omega_N\}=$$

$$P\{x\in R_N/\omega_{INTR}\}P(\omega_{INTR})+P\{x\in R_{INTR}/\omega_N\}P(\omega_N)=$$

$$\int_{-\infty}^{x^*}p(x\mid\omega_{INTR})P(\omega_{INTR})dx+\int_{x^*}^{\infty}p(x\mid\omega_N)P(\omega_N)dx=$$

$$\frac{1}{2}\left[\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{2}\exp\left[-\frac{1}{2}(x-4)^2\right]dx+\int_{2}^{\infty}\frac{1}{\sqrt{2\pi}}\exp\left[-\frac{1}{2}(x)^2\right]dx\right]=$$

$$\frac{1}{2}\left[\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{-2}\exp\left[-\frac{1}{2}(y)^2\right]dy+\int_{2}^{\infty}\frac{1}{\sqrt{2\pi}}\exp\left[-\frac{1}{2}(x)^2\right]dx\right]=$$

$$\frac{1}{2}[0.0228\ +0.0228\ ]=0.0228$$

Example of open question

Consider a two-class problem in $\mathbb{R}^2$ (two-dimensional feature space). Each pattern is characterized by a numerical feature vector in $\mathbb{R}^2$. **x = (x$_1$ x$_2$)**

Each class has a Gaussian probability density function:

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{(2\,\pi)^{d/2}\,|\Sigma_\mathrm{i}|^{1/2}} \exp\left[-\frac{1}{2}\,(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma_\mathrm{i}^{-1}\,(\boldsymbol{x}-\boldsymbol{\mu}_i)\right]$$

With:

$$\mathrm{P}(\omega_1) = \mathrm{P}(\omega_2);\quad \Sigma_1 = \Sigma_1 = \Sigma = \begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} \qquad \boldsymbol{\mu}_1 = \begin{pmatrix}0\\0\end{pmatrix};\ \ \boldsymbol{\mu}_2 = \begin{pmatrix}1\\1\end{pmatrix}$$

<u>Question</u>: Using the MAP (Maximum A Posteriori) decision rule: Classify the pattern x_T=(1/2, 1/3) using the MAP rule by computing explicitly the decision regions and tell to which decision region the pattern x_T=(1/2, 1/3) belongs to.

$$p(\boldsymbol{x}|\omega_i) = \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\,\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2\right]$$

$$p(\boldsymbol{x}|\omega_1) = \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\,\|\boldsymbol{x}-\boldsymbol{\mu}_1\|^2\right] = \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\,[(x_1)^2+(x_2)^2]\right]$$

$$p(\boldsymbol{x}|\omega_2) = \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\,\|\boldsymbol{x}-\boldsymbol{\mu}_2\|^2\right] = \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\left((x_1-\mu_{2,1})^2+(x_2-\mu_{2,2})^2\right)\right] =$$

$$= \frac{1}{2\,\pi}\exp\left[-\frac{1}{2}\,[(x_1)^2+(x_2)^2+2-2\,x_1-2\,x_2]\right]$$

$$p(\boldsymbol{x}|\omega_1) > p(\boldsymbol{x}|\omega_2)$$

$$\exp\left[-\frac{1}{2}\,[(x_1)^2+(x_2)^2]\right] > \exp\left[-\frac{1}{2}\,[(x_1)^2+(x_2)^2]\right]\ \exp\left[-\frac{1}{2}\,[2-2\,x_1-2\,x_2]\right]$$

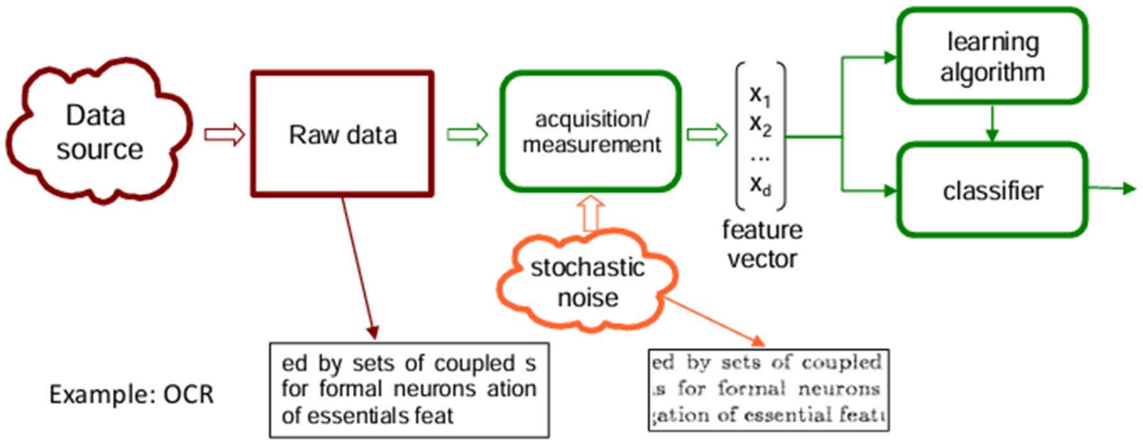
$$1 > \ \exp\left[-\frac{1}{2}\,[2-2\,x_1-2\,x_2]\right]$$

$$0 > \ \ x_1+x_2-1 \Leftrightarrow \ \ x_1+x_2<1 \Leftrightarrow \ \ x_2<1-x_1$$

The pattern x_T=(1/2, 1/3) belongs to class 1.

PART 2

The Classical Statistical Model for Pattern Classification


Data source
Raw data
acquisition/ measurement
x_1 x_2 … x_d
feature vector
learning algorithm
classifier
stochastic noise
Example: OCR
ed by sets of coupled s for formal neurons ation of essentials feat
ed by sets of coupled s for formal neurons ation of essential feat

Note these two implicit assumptions of the model:

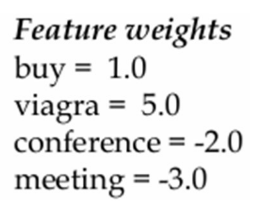
1. The source of data is given, and it <u>does not depend on the classifier</u>
2. Noise affecting data is *stochastic* (randomic)

Can this model be used under "<u>Attack</u>"? Anyone who wanna corrupt my ML system.

NO.

We can think about a **spam filtering classifier.** I can send a "*spam*" email with words such as "*buy, viagra*" just adding good words such as "*meeting, conference*". This is able to corrupt my classifier:

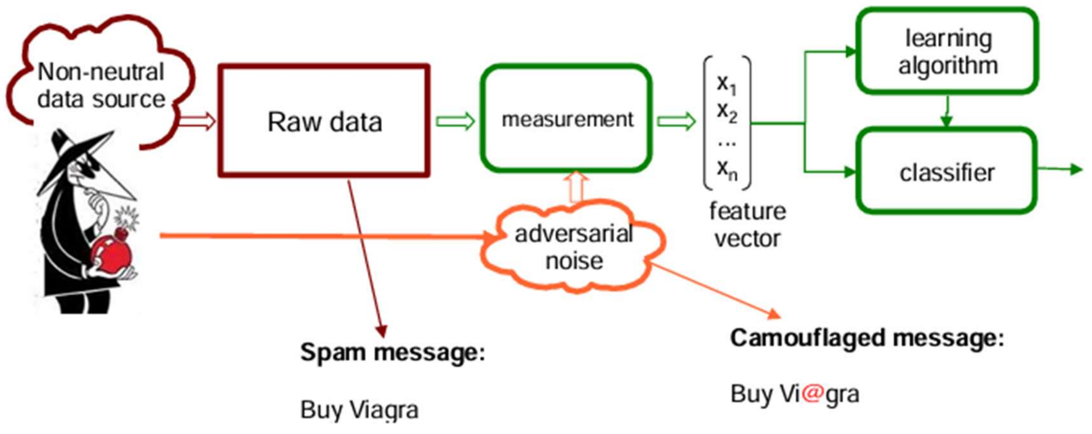
Total score = 1.0 < 5.0 (threshold)



Feature weights	
buy =	1.0
viagra =	5.0
conference =	-2.0
meeting =	-3.0

So, this model **is not good for spam filtering!!** Adversarial tasks are a mission impossible for the classical model.

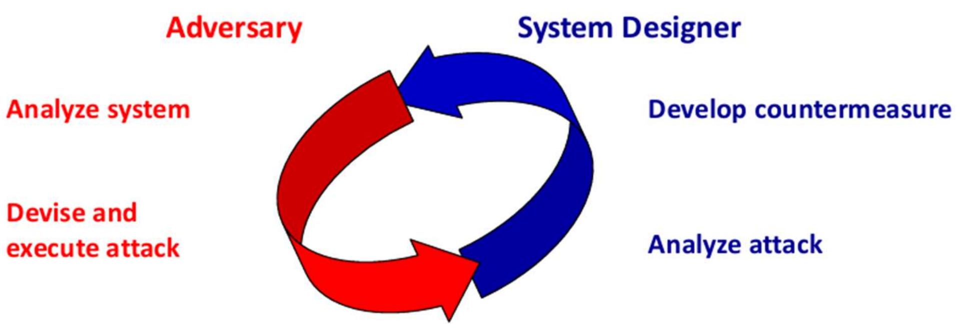
Note that spammers corrupt patterns with a <u>noise that is not random</u>.



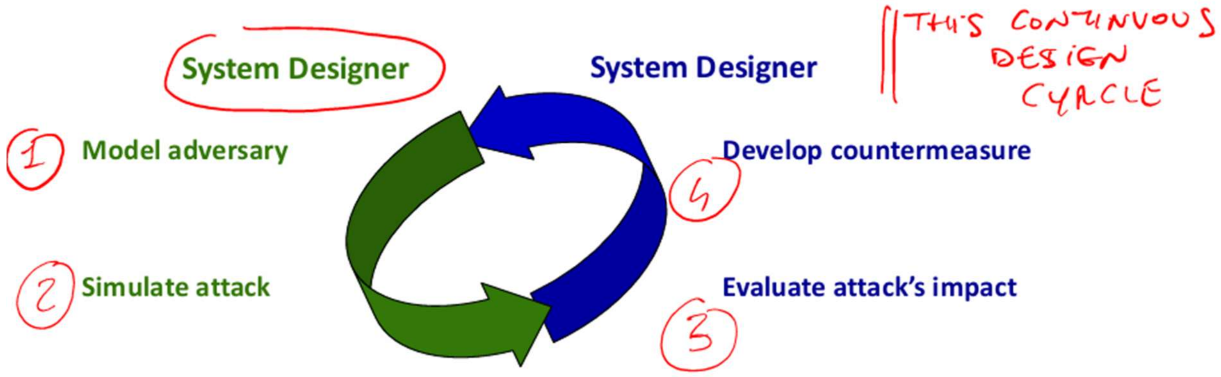
1. The source of data is *not neutral*, it depends on the classifier
2. Noise is not stochastic, it is *<u>adversarial</u>*, crafted to maximize the probability of error

So, how should we design pattern classifiers under attack?
ARM RACE:

1. In 2004 spammers invented a new trick for evading anti-spam filters. As filters did not analyze the content of attached images. Spammers embedded their messages into images...so evading
2. The PRALab team proposed a countermeasure against image spam:
   a. Text embedded in images is read by Optical Character Recognition (OCR)
   b. OCRing image text and combining it with other features extracted from the email data allows discriminating spam/ham emails successfully
3. Spammers reacted quickly with a countermeasure against OCR-based solutions
   a. They applied content-obscuring techniques to images, like done in CAPTCHAs, to make OCR systems ineffective without compromising human readability.
4. PRA Lab did another countermove by devising features which detect the presence of spammers' obfuscation techniques intext images, detecting:
   a. Characters fragmented or mixed with small background components
   b. Characters connected through background components
   c. Non-uniform background, hidden text
   • This solution was deployed as a new SpamAssassin plugin (called *Image Cerberus*)



Machine learning systems should be aware of the ***arms race*** with the adversary.



The Three Golden Rules

1. **Know your adversary**: modeling your adversary
2. **Be proactive**: evaluating the security of your ML algorithm
3. **Protect your classifier**: develop contromeasures defences

"*What if scenario analysis*"

1) Know your adversary

It means one thing: create a model of your adversary: you should clarify the **Goal**, **Knowledge** and **Capability** of my adversary.

The **Goal** of the adversary is an **integrity violation**. Doing misclassification that do not compromise normal system operation. Sometimes the goal is to create an **availability security violation**. Or it can be a **confidentiality/privacy goal**: creating a model of the adversary means making assumptions.

Knowledge could be: 1} *perfect* (white box); 2} *limited* (gray box); 3} *zero* (black box).
Kerckhoffs' Principle states that the security of a system should not rely on unrealistic expectations of secrecy. *Secure systems should make minimal assumptions about what can realistically be kept secret from a potential attacker.*
For machine learning, one could assume that the adversary is aware of the learning algorithm and can obtain some degree of information about the training data. But the best strategy is to assess system security under different levels of adversary's knowledge

Capability

Attackers may manipulate **training** data and/or **test data.**

Training: Influence model at training time to cause subsequent errors at test time *poisoning attacks*, backdoors.
Constraints: maximum number of samples that can be added to the training data. The attacker usually controls only a small fraction of the training samples.

Test: Manipulate malicious samples at test time to cause misclassications evasion attacks, adversarial examples.
Constraints: maximum amount of modifications (e.g., max. number of words that are modified in spam emails).

The design and analysis of a system should avoid unnecessary or unreasonable assumptions on the adversary's capability. Conversely, analysing the capabilities of an omnipotent adversary reveals little about a learning system's behaviour against realistically-constrained attackers. The best strategy is to assess system security under different levels of adversary's capability.

2) **Be Proactive:** Try to anticipate the adversary!

- What is the **optimal attack** the attacker can craft?
- What is the *expected performance decrease* of your pattern classifier?

Evasion Attacks against Machine Learning at Test Time

Goal: evasion

Knowledge: perfect (*white-box* attack)

$$f(x) = \mathrm{sign}(g(x)) = \begin{cases} +1, & \text{malicious} \\ -1, & \text{legitimate} \end{cases}$$

Attack strategy:

$$\boxed{\min_{x'} g(x') \quad \text{s.t.} \ \|x - x'\|_p \leq d_{\max}}$$

d_{max} is the maximum capability of the adversary

Non-linear, constrained optimization: **Projected gradient descent.** Computed with SVMs, Neural Networks.

Why the adversary is interested to MINIMIZE g(x)? The goal is to minimize g(x) because f(x) will be negative (legitimate), so it will evade the classifier.

PART 3.1

Gradient descent is an algorithm used for minimizing a **Loss Function.** We want to find the minimum of the Loss Function with respect to the *parameter vector.*

The key idea of *gradient-based optimization* is to start from a **random point** in the **parameter space** and then iteratively update the parameters along the gradient direction.

The simplest gradient-based optimizer is the **steepest-descent** method

```
1. initialize $\theta$, $\eta$, $K$, $\varepsilon$
2. for k in {0, 1, ... , $K-1$}:
3.     $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta \nabla L(\boldsymbol{\theta}_k)$
4.     if $|L(\boldsymbol{\theta}_k) - L(\boldsymbol{\theta}_{k+1})| < \varepsilon$:
5.         break
```

η is the step, called also **learning rate**

if it is too large, I can risk to don't converge

if it is too small, convergence is too slow

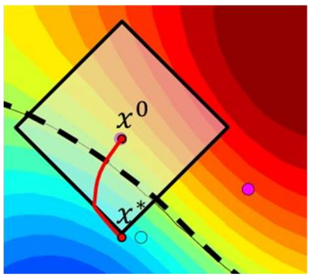
Idea: Just *decrease* η_k at each iteration, with different *decaying rates*

Projected Gradient Descent

Gradient descent minimizes a function by moving in the negative gradient direction at each step. There is no constraint on the variable.

Problem 1: $\min_x f(x)$ $\qquad x_{k+1} = x_k - t_k \nabla f(x_k)$

Projected gradient descent minimizes a function subject to a constraint. At each step we move in the direction of the negative gradient, and then "project" onto the feasible set. It means that if the solution is out of the square in the image (that is the constraint), I project it inside the square, in order to satisfy the constraint.

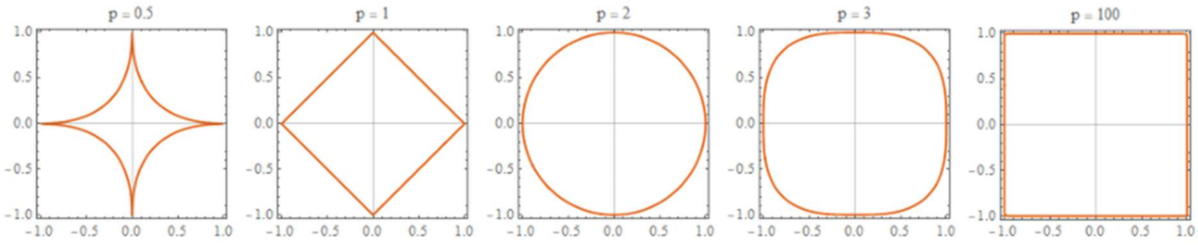


Problem 2: $\min_x f(x)$ subject to $x \in C$

$$y_{k+1} = x_k - t_k \nabla f(x_k)$$
$$x_{k+1} = \arg\min_{x \in C} \|y_{k+1} - x\|$$

Effect of Different Norms

Depending on the **choice of the norm**, we have to consider different evasion attacks.
Which kind of norms I have? L_0, L_1, L_2, …., L_∞
L_2 is the "*Quadratic norm*" also called "*Eucledian Distance*" that is a cirlce

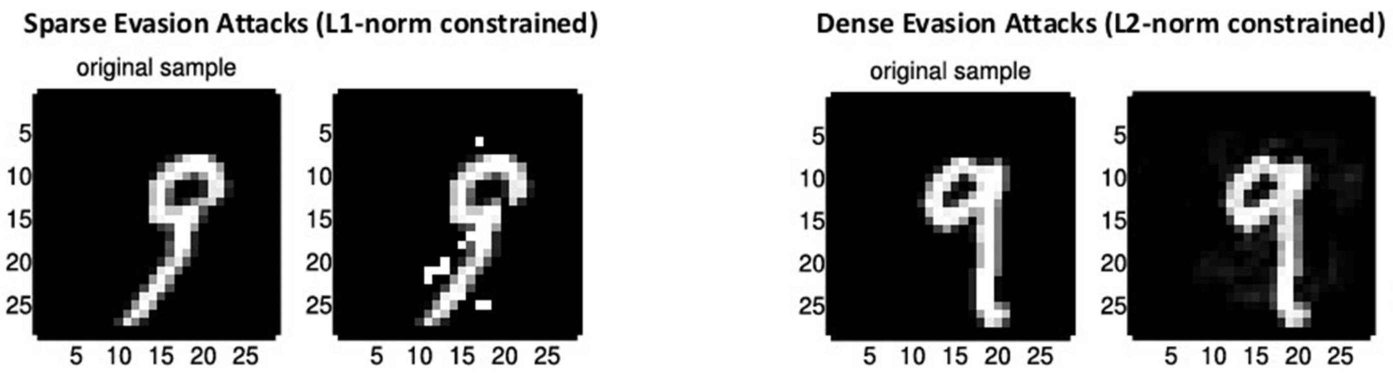


Depending on the choice you have a very different effect on the mathematical problem of the evasion attack. **Why?** Because it is related to the effect of **sparsity.**
SPARSITY: A lot of the values of the weights **w_s** are equal to zero **(w_s = 0)**

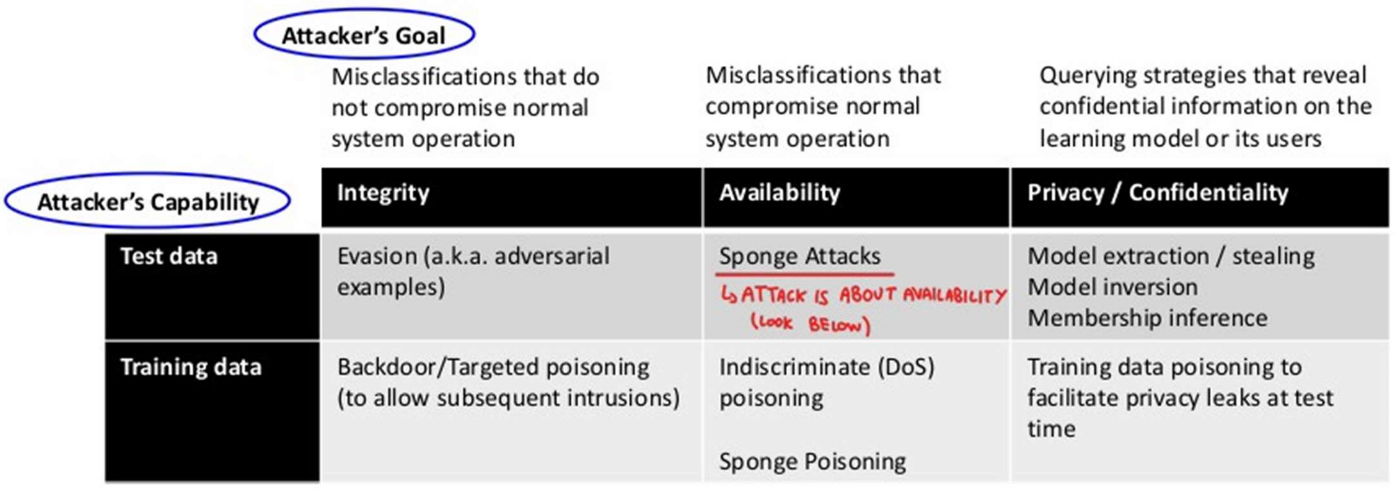
X = [1 2 3 3 7 2 3] is not sparse $\qquad$ X = [0 0 0 1 0 0 1] is sparse

Sparsity is good for a ML model because it means *less computation*.

Sparse Attack: better for spam filtering. (It perturbates only few features of the image)
Dense Attack: better for image recognition. (It perturbates a little bit all the image).


Sparse Evasion Attacks (L1-norm constrained)
original sample
Dense Evasion Attacks (L2-norm constrained)
original sample

ATTACKS AGAINST MACHINE LEARNING



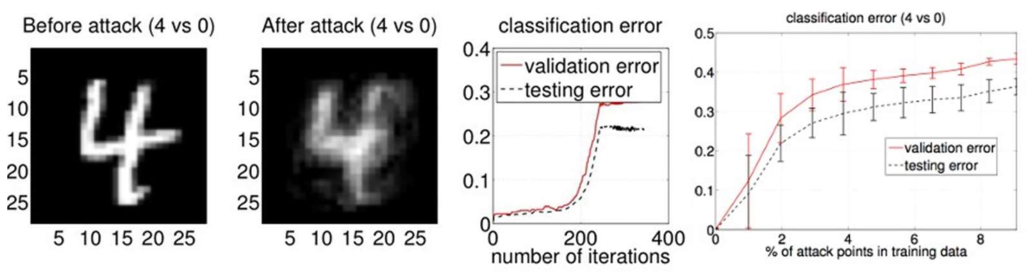
Attacker's Capability \ Attacker's Goal	Integrity	Availability	Privacy / Confidentiality
	Misclassifications that do not compromise normal system operation	Misclassifications that compromise normal system operation	Querying strategies that reveal confidential information on the learning model or its users
Test data	Evasion (a.k.a. adversarial examples)	Sponge Attacks ↳ ATTACK IS ABOUT AVAILABILITY (LOOK BELOW)	Model extraction / stealing Model inversion Membership inference
Training data	Backdoor/Targeted poisoning (to allow subsequent intrusions)	Indiscriminate (DoS) poisoning Sponge Poisoning	Training data poisoning to facilitate privacy leaks at test time

<u>Attacker's Knowledge</u>: white-box / gray-box / black-box (query/transfer) attacks (transferability with surrogate learning models)

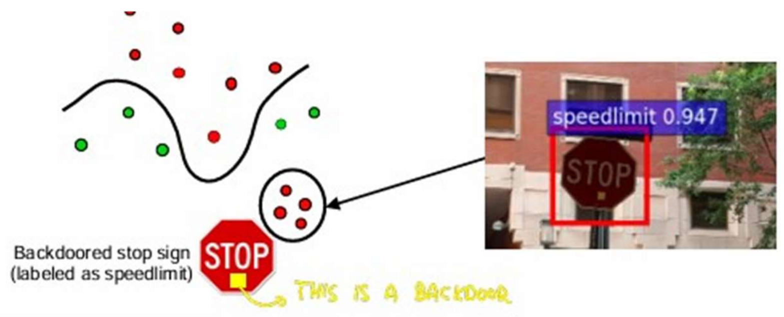
1) **Sponge Attack** is an attack created to pertubate input data so that the *energy consumption* of a given ML model increase a lot **(at test time).** *Energy consumption* of a ML model depends on how much computation it needs to do.

2) Or also **at training attack,** called **Sponge Poisoning.** *Poisoning* refers to all the attacks affetcting the training data. So, it is a sponge attack, but manipulating the training data.

3) An **Indiscriminate Poisoning Attacks** *inject* few training points to cause large testing error (on clean samples).



4) **Backdoor Poisoning Attacks** place mislabeled training points in a region of the feature space far from the rest of training data. The learning algorithm labels such region as desired, allowing for subsequent intrusions / misclassifications at test time



5) Membership Inference Attacks (privacy attack)

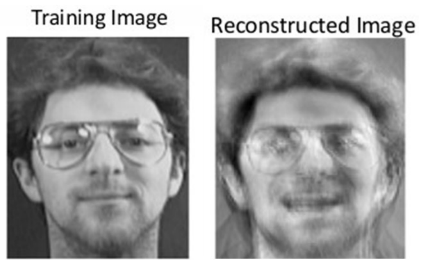
Goal: to identify whether an input sample is part of the training set used to learn a deep neural network based on the observed prediction scores for each class. So:
"*Was this specific data record part of the training set?*"

6) Model Stealing/Extraction Attacks (IPR) (privacy attack)

The attacker buys the product and he starts doing experiments with it, trying to discover the weights $\mathbf{w}_s$ and he tries to build is own model (stolen).

7) Model Inversion Attacks (privacy attack)

I want to discover which training samples have been used for example to reconstruct the same example.



Why Is AI Vulnerable?

- Underlying assumption: past data is representative of future data (IID data)
  - IID (Identically indipendent distributed).
- Adversarial attacks point exactly at this lack of robustness which comes from IID specialization

Why Is AI Security and Safety an Important Concern?

Safety: How much ML can work well when test data are different then training data
Security: how much ML works againts evasion attacks.

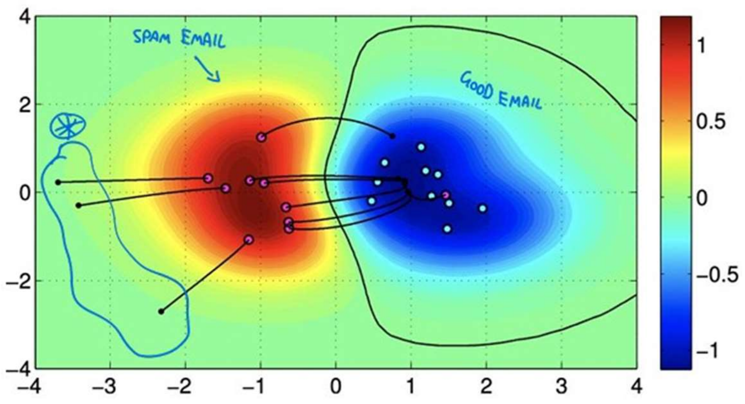
The **Explainability** means how can we understand if our ML model works in a way is reasonable.

A **Spurious Correlation** means that we want to explain which are the most important part of an image that ML system takes in account for image recognition. In order to do that we need ***explainability algorithms***.
(EXAMPLE: if I recognize a dog as a wolf, but through a *explainability algorithms* I find out that the most important part of the image is the background, it is a _Spurious Correlation_ that is not good).

Problem: Do We Always Evade with Gradient Descent?

NO! Look at the following figure (Spam Filtering example): Many red samples do not cross the boundary, even if they are able to get sufficiently far from the red class.

If we use only the *gradient descent* to create an evasion attack, it may happen that we find a solution ⊛ that is just a <u>mathematical soulution</u> and it does not look like a *good email* (good emails are in the blue region).



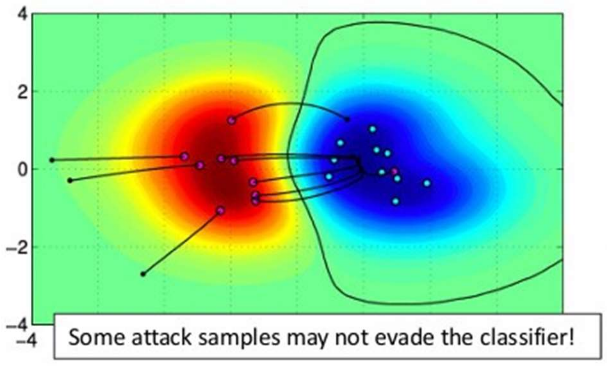
So, depending on the threshold for filtering, the solutions inside ⊛ **may or may not evade the classifier.**

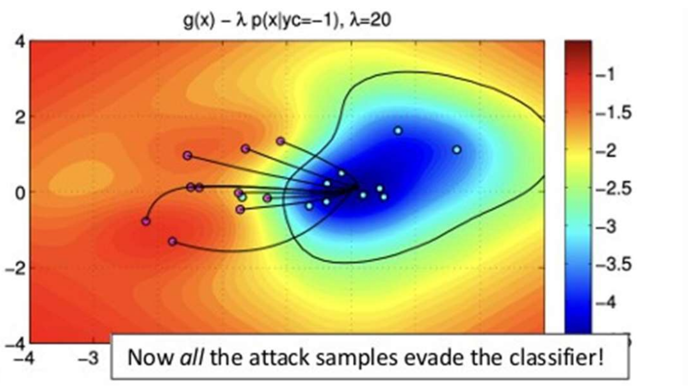
It means that with an adversarial noise, I'm able to evade the classifier (going far from the red area), but sometimes I create a solution that is not a good example. Not all the gradient-descent directions that minimize the function are good for evasion, we want to go in the direction of the "target" class (blue area). So, I need to add an **additional term**, in order to create the so called **Mimicry Attack** (going inside the blue region).

Solution: Kernel Density Estimation (KDE)

$$\boxed{\begin{aligned}&\min_{x'}\{g(x') - \lambda p(x'|y_t)\}\\ &\text{s.t. } \|x - x'\|_p \leq d_{\max}\end{aligned}}$$

- ***p*(x'|y$_t$):** probability that x after the perturbation (that is x') is *inside* the blue region. I can compute it <u>if I have a training set</u>.
- **λ** is the *evasion rate*.





***p*(x'|y$_t$)** is also called **Density Estimation.** We can use different methods for density estimation p(x/y): *Knn density estimation, Parzen windows,...*

PART 3.2

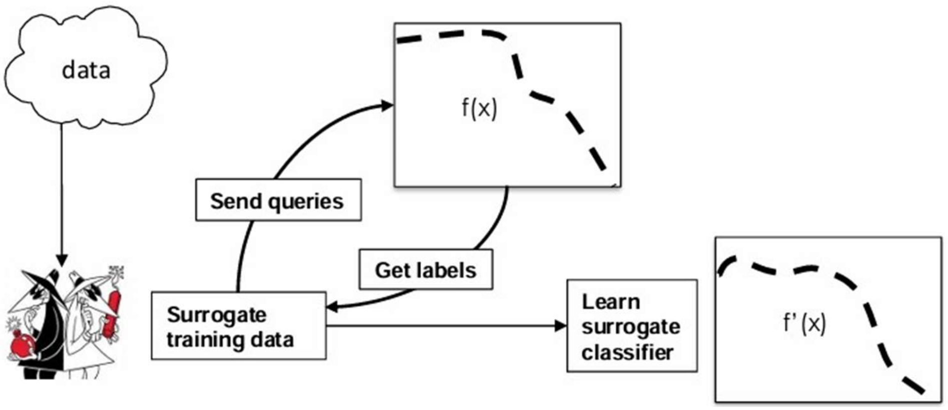
White-box to Grey-box Attacks: Basic Concepts

White-box: Complete knowledge about the ML system to attack, I can do what i want

Gray-box: Partial knowledge about the ML system to attack

What does the adversary do in the case of Limited Knowledge?

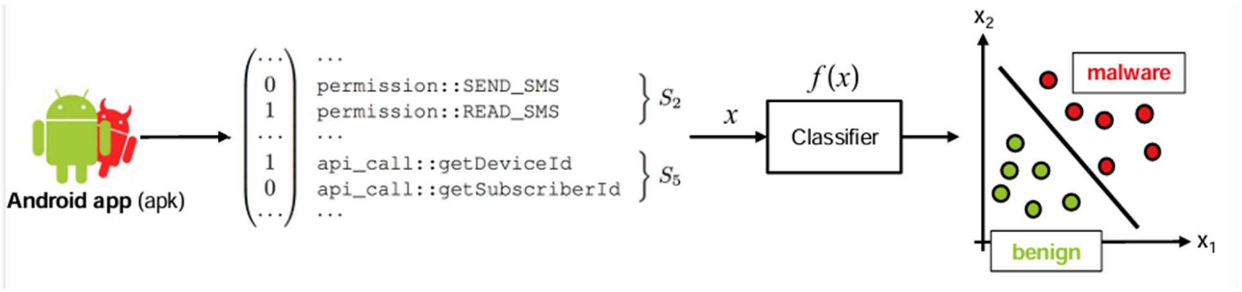
- Only feature representation and (possibly) learning algorithm are known
- Surrogate data sampled from the same distribution as the classifier's training data
- Classifier's feedback to label surrogate data



Then, I can try to attack the estimated model *f'*(x), if it works I can try to attack the real model *f*(x).

Experiments on Android Malware Detection

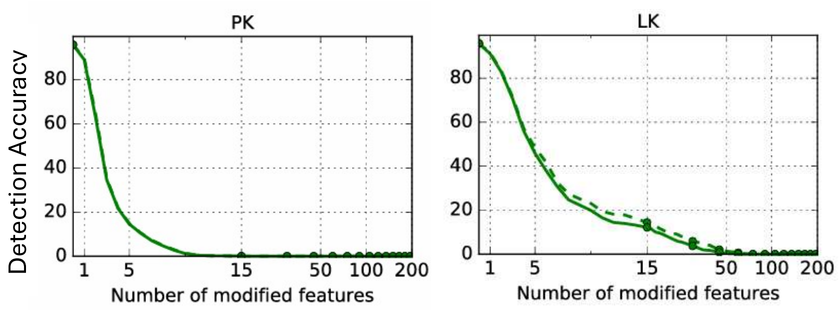
- Android malware detection *directly on the mobile phone*
- Linear SVM trained on features extracted from static code analysis



A <u>binary feature vector</u> is extracted from the android app.

The classifier is a ***Linear** ML Model*. It is linear because it should be fast and it shouldn't need a lot of computational time.

Trying to evade the ML Detector Model

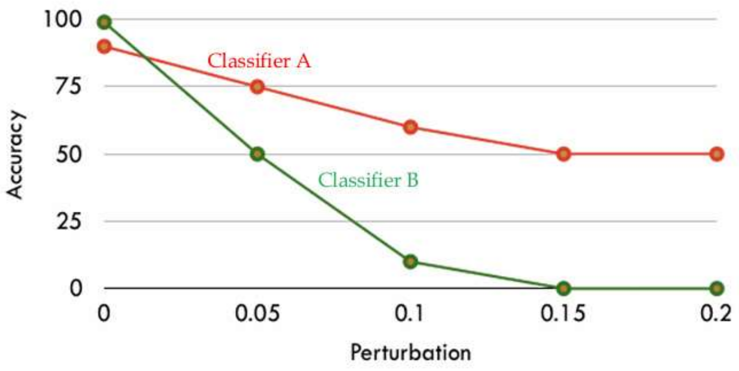


PK: Perfect Knowledge (white-box)

LK: Limited Knowledge (gray-box)

(Detection Accuracy equal to Zero, means that the ML detector is complitely evaded)
Number of modified features is the ***capability*** of the attacker.

These 2 graphs are related to the concept of **Security Evaluation Curves** (fig below)**.** It is important to understand how many features I have to modify in order to evade the classifier.



I use the **Security Evaluation Curves** to compare different ML Model.

In this case, **Classifier A** is better **!!**

"*Perturbation*" is the number of modified features, it depends on the Adversary Model (in particular on the *capability*).

Minimum-norm vs Maximum-confidence Evasion Attacks

There are 2 categories of evasion attack: **minimun-norm (distance)** and **maximum-confidence** attack.

1) **Minimum-norm:** I want to find the smallest perturbation needed to evade the ML model.

2) **Maximum-confidence:** Is not the smallest perturbation needed as over. Instead, to find the evasion attack exploiting the *maximum capability* that I have. It is called *maximum-confidence* because it has a quite large probability of being succesfull, even if the *knowledge* is not perfect.

If I have a *partial knowledge*, the **maximum-confidence attack** has more chances of being succesfull. It is better if I exploit the full capability that I have..
If I have *perfect knowledge*, the **minimum-norm attack** is better.

General Categorization

In general, evasion attacks, can be represented in a mathematical point of view as a problem of minimization:

$$\min_{\boldsymbol{\delta}}[L(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}), \|\boldsymbol{\delta}\|_p]$$

$\boldsymbol{\delta}$ is the perturbation;

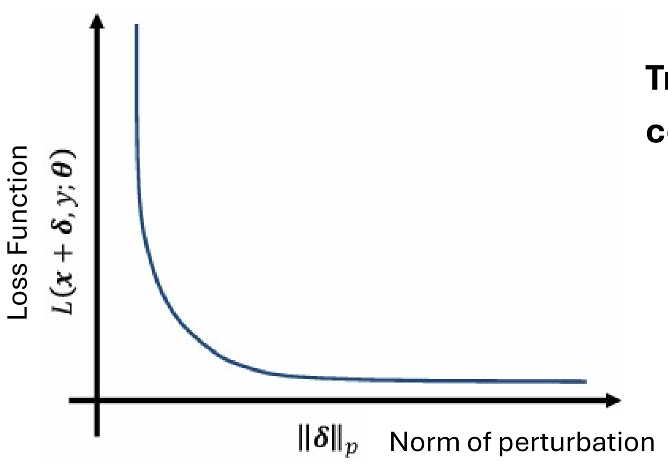
$||\boldsymbol{\delta}||_p$ is the norm of the perturbation. ($p = 1,2,\ldots,\infty$)

L is the Loss Function that I use for the *trainining of my ML model*

Y is the target label

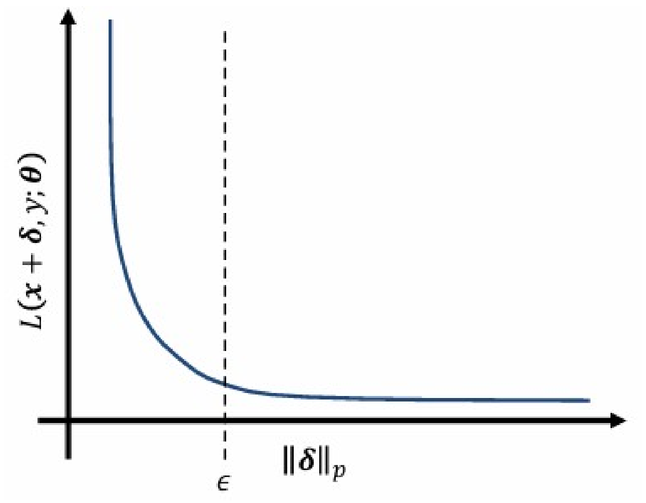
θ is parameter vector

Pareto Frontier



Trade-off between **misclassification confidence** and **perturbation size**

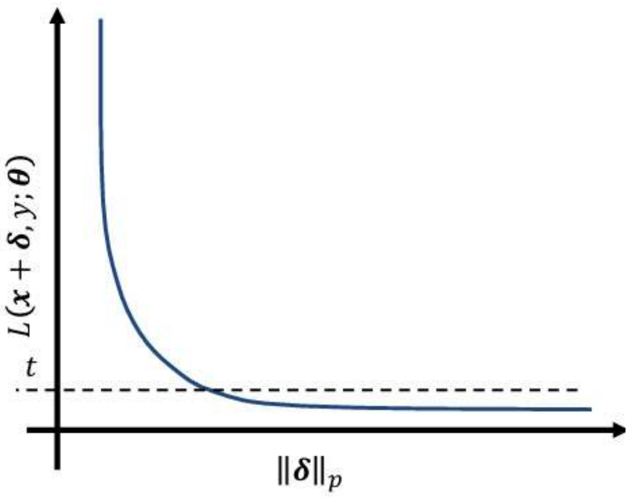
Maximum-confidence Attacks, constraint



$$\min \; L(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}), \quad \text{s.t.} \; \|\boldsymbol{\delta}\|_p \le \epsilon$$

We have an HARD contraint in term of perturbation **ε** and we want to minimize the Loss Function **L.**

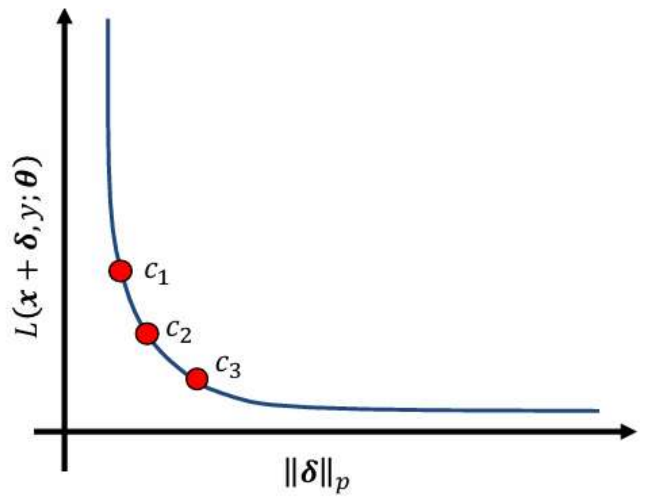
Minimum-norm Attacks, constraint



$$\min \; \|\boldsymbol{\delta}\|_p \quad \text{s.t.} \; L(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) < t$$

We have an HARD contraint on the Loss Function **L** and we try to minimize the perturbation **ε.**

Soft-constraint: Trade-off Solution



All constraints are imposed as quantities modulated by coefficients, behaving as regularizers.

Modulating the multipliers shifts the solution towards trade-off between score and distance

$$\min L(x + \delta, y; \theta) + c\|\delta\|_p$$

C is a **penalty term.**

From Binary to Multiclass Evasion

In multiclass problems, classification errors occur in different classes.

Thus, the attacker may aim to have:

1) A sample misclassified as any class different from the true class: **error-generic (indiscriminate) attacks**
2) A sample misclassified as a specific class: **error-specific (targeted) attacks**

1) Error-generic (indiscriminate) attacks

- **k** is the true class (**blue**)
- **l** is the competing (closest) class in feature space (**red**)

$$\Omega(\boldsymbol{x}) = f_k(\boldsymbol{x}) - \max_{l \neq k} f_l(\boldsymbol{x})$$

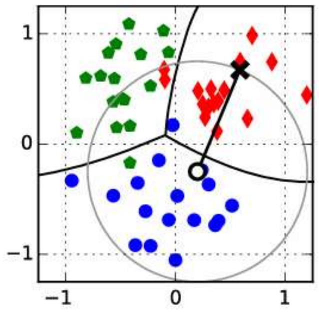
(l≠k means any class that is not k)

The attack **minimizes** the objective to have the sample misclassified as the closest class (could be any!)

$$\begin{aligned} \min_{\boldsymbol{x}'} \quad & \Omega(\boldsymbol{x}') \,, \\ \text{s.t.} \quad & d(\boldsymbol{x}, \boldsymbol{x}') \leq d_{\max} \,, \\ & \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{\text{ub}} \,, \end{aligned}$$

Is the *capability* (circle)

This is something that depends on the application. (For instance, talking about a picture, the value of the pixel must be in the range [0, 255]



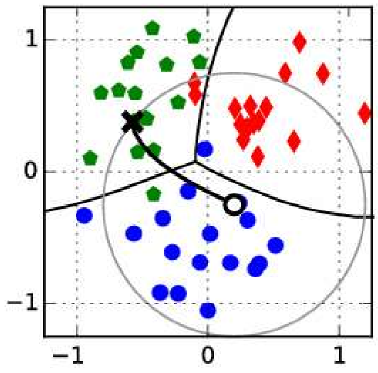
2) Error-specific (Targeted) Evasion

- **k** is the true class (green)
- **l** is the competing class (initially the **blue** class)

$$\Omega(\boldsymbol{x}) = f_k(\boldsymbol{x}) - \max_{l \neq k} f_l(\boldsymbol{x})$$

I want to find the maximum because the value of the target class has to has to have an high value. The other classes a low value (riscolta bene)
The attack **maximizes** the objective to have the sample misclassified as the target class

$$\begin{aligned} \max_{\boldsymbol{x}'} \quad & \Omega(\boldsymbol{x}') , \\ \text{s.t.} \quad & d(\boldsymbol{x}, \boldsymbol{x}') \leq d_{\max} , \\ & \boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{\text{ub}} , \end{aligned}$$



All the things that we have written with the function **f(x)** can be rewritten with the **Loss Function L**.
We can formalize/generalize adversarial attacks as:

- $\max_{\boldsymbol{\delta}} L(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta}) = \min_{\boldsymbol{\delta}} \; -L(\boldsymbol{x}+\boldsymbol{\delta}, y, \boldsymbol{\theta})$, for indiscriminate attacks. The loss value for the true label y is maximized so that is maximized the probability to have x classified as belonging to a different class;
- $\min_{\boldsymbol{\delta}} \; L(\boldsymbol{x}+\boldsymbol{\delta}, y_t, \boldsymbol{\theta})$, for targeted attacks, with $yt \neq y$. The loss value is minimized for $yt \neq y$ so that is maximized the probability to have x classified as belonging to the class yt

Maximum-confidence Attacks

Which means I exploit all the capability to execute the attack with maximum confidence, with minimum norm on the other hand I find the smallest perturbation to change the recognition result.

1) Fast Gradient Sign Method (2015)

Perturbed image obtained as: $\boldsymbol{x}^{\star} = \boldsymbol{x} + \epsilon \,\text{sign}(\nabla L(\boldsymbol{x}, y; \boldsymbol{\theta}))$

Assumption: my ML model can be approximated to a *Linear Model*. First order Taylor expansion.

I add the perturbation multiplied by the sign of the Gradient if the Loss Function → it follows the direction of the gradient.

It is maximum confidence because I take in account my capability, as a maximum confidence attack, we can exploit all the capability.

It is very fast → very easy to implement.

$$\max_{||\boldsymbol{\delta}||_\infty \le \epsilon} L(\boldsymbol{x}+\boldsymbol{\delta}, y; \boldsymbol{\theta}) \approx \quad \text{(linear approximation)}$$

$$\max_{||\boldsymbol{\delta}||_\infty \le \epsilon} L(\boldsymbol{x}, y; \boldsymbol{\theta}) + \boldsymbol{\delta}^{\mathrm{T}} \nabla L(\boldsymbol{x}, y; \boldsymbol{\theta}) =$$

$$L(\boldsymbol{x}, y; \boldsymbol{\theta}) + \max_{||\boldsymbol{\delta}||_\infty \le \epsilon} \boldsymbol{\delta}^{\mathrm{T}} \boldsymbol{g}$$

The solution is to set

$$\boldsymbol{\delta}^\star = \epsilon \, \mathrm{sign}(\boldsymbol{g})$$

- g stands for the gradient.
- The effectiveness of the attack depends on how good the linear approximation of the ML system is.
- Depending on the norm we use we obtain different results.

Why Are Attacks Effective / Imperceptible against DNNs?

Let's pretend that a deep network behaves in a linear way...

Under this linearity assumption, the output of the net to the adversarial input

$$\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\delta} \quad \text{is} \quad \mathbf{w}^{\mathrm{T}} \tilde{\boldsymbol{x}} = \mathbf{w}^{\mathrm{T}} \boldsymbol{x} + \mathbf{w}^{\mathrm{T}} \boldsymbol{\delta}$$

The key concept is that **if the dimensionality of the input $\boldsymbol{x}$** is very high and the vector of the adversarial perturbation $\boldsymbol{\delta}$ is aligned with the "classifier" (its weight vector $\mathbf{w}$), then **many infinitesimal changes to the input** add up to **one large change to the output,** and evasion attacks can result "invisible"**.**

Projected Gradient Descent

Is the iterative version of FGM → fast gradient signed method. Also this method depends on how good of a linear approximation I can get of our ML model. I usually use this method when we don't obtain good results with **Fast gradient signed method.**

Minimum-norm Attacks

As said before, I find the smallest perturbation to change the recognition result. It's a minimum norm attack. I don't take into consideration the capability.

1) Adversarial Examples against DNNs

<u>Constraint</u>: Final recognition is different than the Initial recognition (being succesfull)

$$\min_{\boldsymbol{\delta}} ||\boldsymbol{\delta}||_2$$
$$\text{s.t.} \quad f(\boldsymbol{x}+\boldsymbol{\delta}) = y_t \qquad (y_t \neq y)$$
$$\mathbf{x} + \boldsymbol{\delta} \in [0,1]^{\mathrm{d}}$$

Constraint:

$$\mathbf{x} + \boldsymbol{\delta} \in [0,1]^{\mathrm{d}}$$

The attack stops when the final recognition is different from the right recognition and the evasion attack is said to be successful.

We introduce the constraint in the minimization problem in this way → Relaxation method:

$$\min_{x+\delta\in[0,1]^d} c\cdot||\boldsymbol{\delta}||_2 + L(\boldsymbol{x}+\boldsymbol{\delta}, y_t, \boldsymbol{\theta})$$

- C is a parameter that we need to tune up, and it changes how much we consider the perturbation in the function.

2) DeepFool

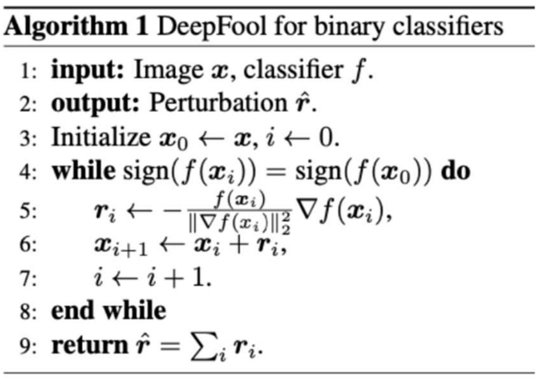
It's a minimum norm evasion attack. Again it's based on the linear approximation of f().
Basic idea: for a linear binary classifier, the *robustness* of f() at point x_0, is equal to the distance from x_0 to the separating hyperplane. We can compute the distance to the hyperplane and move the point just a little bit over the boundary.
The distance to the hyperplane can be computed in the following way:

$$\text{Distance of } \boldsymbol{x} \text{ to the hyperplane: } \frac{|f(\boldsymbol{x})|}{||\boldsymbol{w}||}$$

Once we know this distance, we have a simple way to compute the adversarial example:

$$\boldsymbol{x}^\star = \boldsymbol{x} - \frac{f(\boldsymbol{x})}{||\boldsymbol{w}||}\boldsymbol{w}$$

If we apply these steps iteratively, we can find an adversarial example that completely changes the classification result.



Algorithm 1 DeepFool for binary classifiers

```
1: input: Image $\boldsymbol{x}$, classifier $f$.
2: output: Perturbation $\hat{\boldsymbol{r}}$.
3: Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
4: while $\operatorname{sign}(f(\boldsymbol{x}_i)) = \operatorname{sign}(f(\boldsymbol{x}_0))$ do
5:     $\boldsymbol{r}_i \leftarrow -\frac{f(\boldsymbol{x}_i)}{\|\nabla f(\boldsymbol{x}_i)\|_2^2}\nabla f(\boldsymbol{x}_i)$,
6:     $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$,
7:     $i \leftarrow i + 1$.
8: end while
9: return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$.
```

DeepFool algorithm for binary classification.

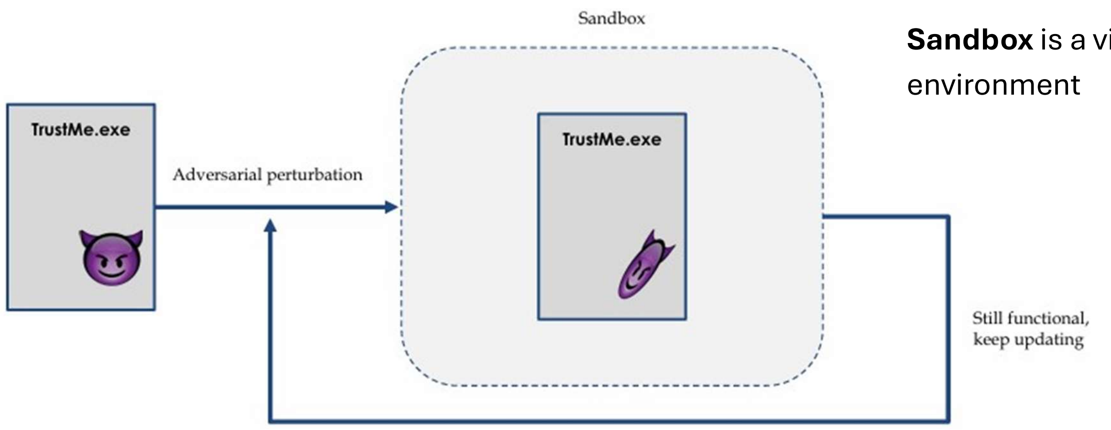
PART 3.3

Adversarial EXEmples: Evading Windows Malware Detectors

Companies claim to use machine learning technologies inside their detectors to spot Windows malware by learning patterns from data.
Deep networks learn "signatures", and they can spot **variants of the same malware.**
But **Machine Learning systems are vulnerable**, they all might be target of evasion attacks!

But How?
Goal: Have a malware that works and that evades antivirus system.
Images and programs are different. If I had an adversarial noise to an image I can evade the ML system for image recognition. **But if I had an adversarial noise without thinking to a malware to evade an antivirus, the file.exe might be not runnable anymore!!!**

We **cannot** use any **Gradient Descent algorithms** for malware. We want to preserve the origin functionality.



Sandbox is a virtual environment

How to bridge these gaps?

1) Formulate the minimization problem differently (we don't have PGD)
2) Study the format that represent programs
    a. We want to create a manipulation that takes the domain knowledge. Then, I know what I have to do if I want to manipulate the system
3) Understand how to exploit the format
4) Chose how to inject or perturb the content
    a. How to create an optimization algorithm (so it is the delta δ)

1) FORMULATION OF THE PROBLEM

Adversarial Attacks For Images

$$\min_{\delta} L(\boxed{f(\phi}(\boxed{h(x;\delta)}), y)$$

Network architecture in the loss
All the internals of a neural network model are hidden inside the loss function

Additive Manipulation
Input samples are injected with additive noise, without any concern on the structure of the file

Adversarial Attacks For Security Detectors

$$\boxed{\min_{\delta}} L(\boxed{f(\phi}(\boxed{h(x;\delta)}), y)$$

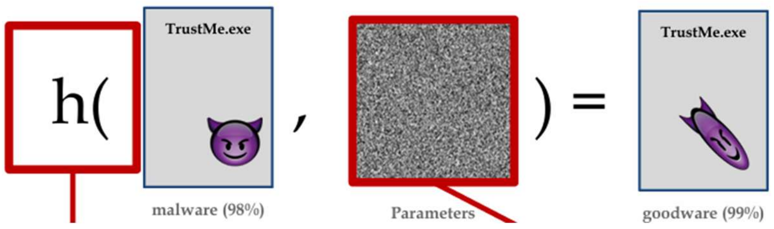
Define the Optimizer
Depending on the differentiability of the compontens, pick a gradient based or gradient-free algorithm

Model function and features
Need to explicit the model function and the features, since they might be non differentiable

Practical Manipulations
No additions, but a complex function that handles format specification by design

- First of all we remove from ***L*** the notion of the model, it is just a *Loss Function*. So, now ***f*** is the model, **Φ** is the *feature extractor*.
- **$h(x;\delta)$:** Generic function that we have to create, it depends on the *domain* that we want to use to apply this manipulation. So, it depends on the format and ambiguities of the model that we want to attack. **X** is the original malware, δ is the noise that we want to inject.

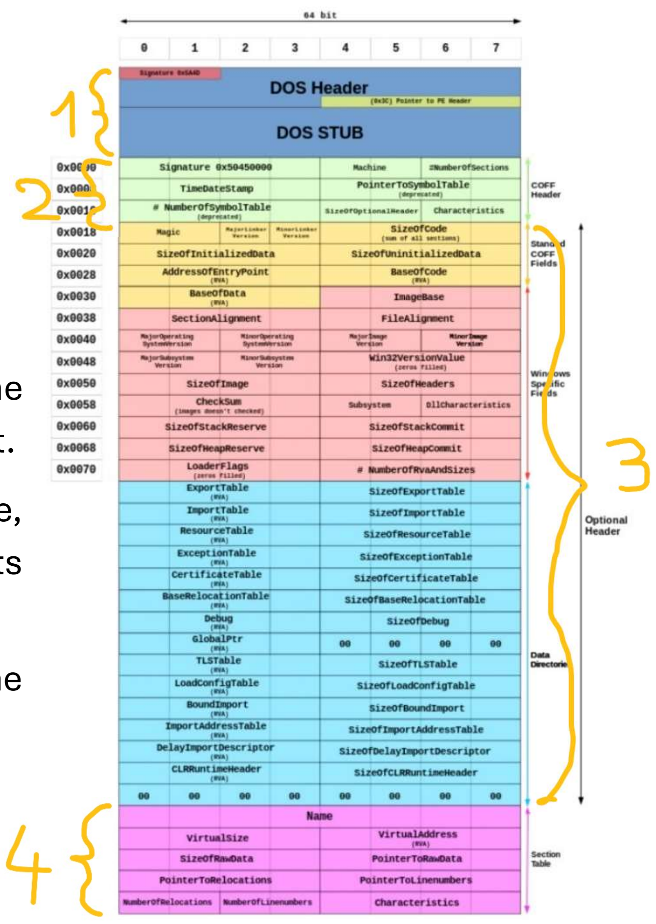
So, we need to define $h\ (x\ ;\ \delta)$ such that:



Windows PE File Format

Format adapted for "modern" programs.
(Before there were other formats, one is the DOS).

DOS is the first OS from Microsoft.



1) DOS Header + Stub

Metadata for DOS program
Executing a modern program in DOS will trigger the "*This program cannot be run in DOS mode*" output.

This header makes the file an MS-DOS executable, so when it's loaded on MS-DOS the DOS stub gets executed instead of the actual program.

The yellow part tells the loader where it can find the *PE Header* (2)

2) PE Header

Real metadata of the program
Describes general information of the file

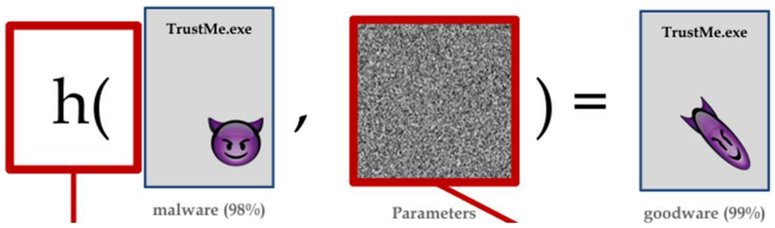
3) Optional Header

Spoiler: not optional at all. It is crucial
Instructs the loader where to find each object inside the file.
If I misplace one Byte here, the program will be corrupted 100%.
In the light-blue part there are the "*tables*" that specify to the loader which are the function needed by the program to work.

4) Section Table and Sections

The "*Section Table*" describes where to find code, initialized data, resources, etc to the loader.
The "Sections" are exactly the pieces of my program.

So, we need to define $h\,(x\,;\delta)$ such that:



There two ways:

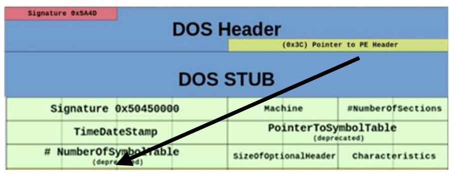
1. **Structural Manipulations**: You DON'T touch the code of the program, but just the MetaData
    a. **Inject the content** (create some spaces in the program by displacing some bytes or adding sections)
    b. **Replacing content**: alter bytes that are not considered at runtime, so that are useless (for instance the dos part (1) is useless)
2. **Behavioral Manipulations** (more difficult): you touch the code, so you can open the functionalities and add new flows of execution
    a. **Packing and obfuscation**: Encrypt your program inside another one. It means that you create another program where you can include also my program and then execute it (such as a matrioska).
    b. **Inject new execution flows**: I do not complicate the program, but I add the execution flows. How? When the loader starts the program, I can tell him to starts from the flow I added.

DOS header perturbations

The attacker edit as much bytes as they want in this part because it is useless. Content loaded in memory, not executed.

<u>Untouched</u>: magic number MZ and offset to real PE header

Between, (1) and (2) there is an offset (that is the *DOS STUB*), if I want I can extend this offset **incrementing** the value of the **PE header** (yellow).



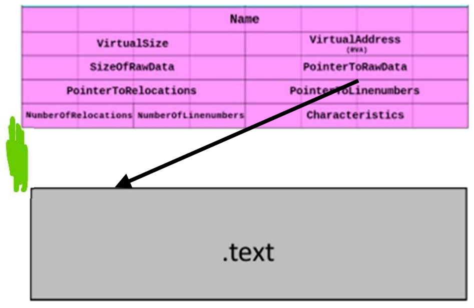
Then, I can insert arbitrary content between DOS header and PE header (not executed)

Content Shifting

Here I can do the same game of before.
Exploit offset in section entry, increment to manipulate the loader in searching for section content.

The attacker can inject content after the section table, or between sections

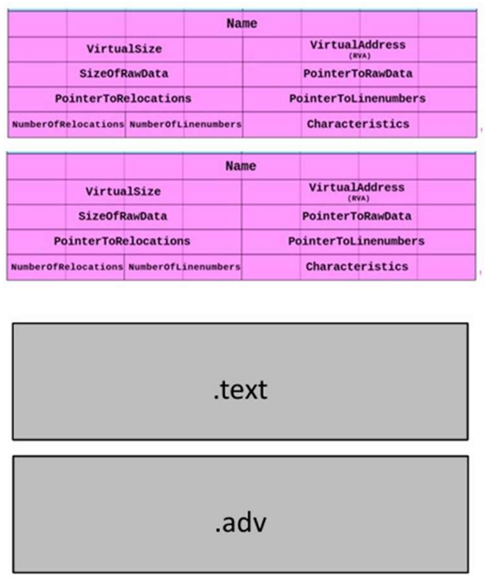
NOT LOADED IN MEMORY, skipped by the loader everything in the green part.



Section Injection

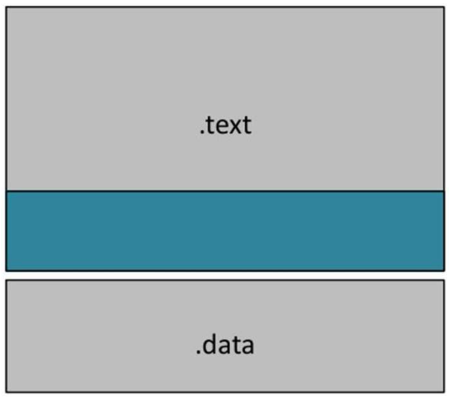
I manipulate the section table to add a new section entry (the second purple table) and I append some content in the end (chunk of bytes, referenced by newly added entry).

I don’t know it will be executed or not



Slack Space

Section content is padded with 0 to keep file alignments.
The compiler puts some 0 at the end for being optimal in loading.
This space is not used by the program and the attacker can rewrite such *slack space*.
Loaded in memory, not executed.

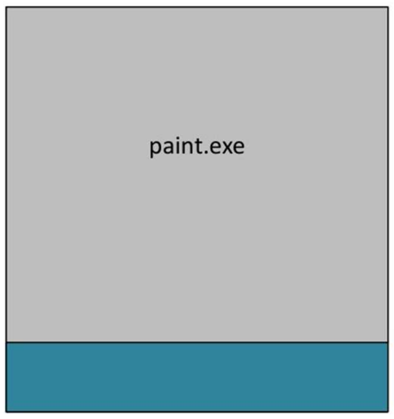


Padding (most simple)

Just appending content at the end, nothing else.

Most trivial manipulation.

Not loaded in memory



Optimization Algorithms

In these malwares there is not metric of distance (for optimitazion).

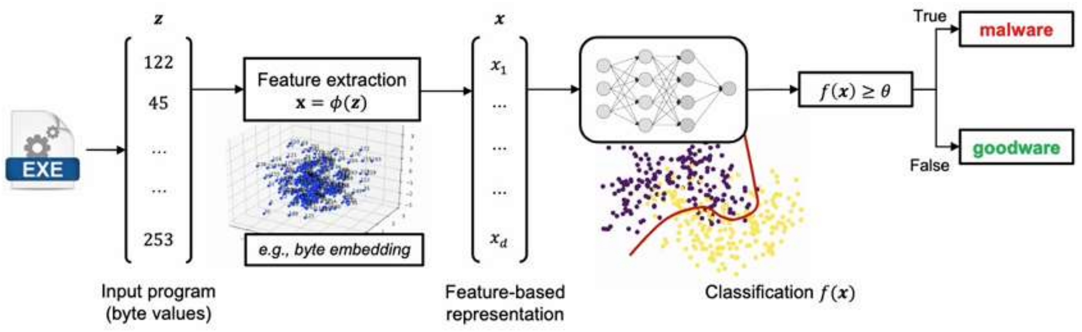
Chosing the strategy accordingly:

1) **Gradient-based:**
    - Own the model AND Model is differentiable
    - YOU CANNOT COMPUTE THE GRADIENT OF WINDOWS DEFENDER: IT IS NOT DIFFERENTIABLE.
2) **Gradient-free:**
    - Model not accessible OR Model is not differentiable

1) Gradient-based strategies

For optimization problems, the first layer of Neural Network acts as a Feature Extractor.

Bytes do not have a distance metric, a feature extractor is **ALWAYS** needed to compute something meaningful.

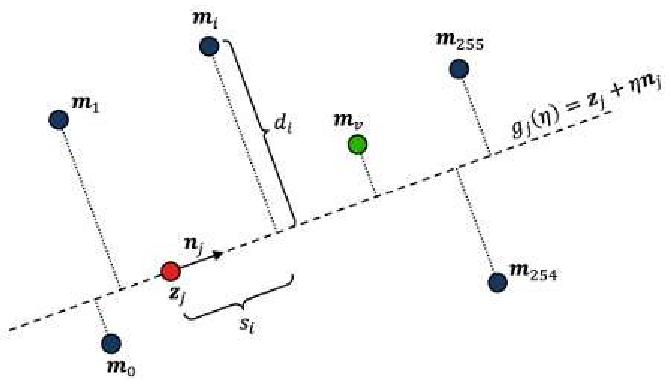


The **Byte Embedding**: We have to compute "*distances*" between our input samples, but it is not possible with bytes. The **Byte Embedding layer** produce a vector for each byte in the input and map it in a 3D space (4D,5D,...or ND space) and computes this sort of "*distances*" between bytes.
The *Embedded Layer* is **not differentiable.**

How to propagate gradient information?

$$\underbrace{\boxed{\frac{\partial L}{\partial \delta}}}_{\text{End-to-end gradient you would like to compute}} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial \phi}\underbrace{\boxed{\frac{\partial \phi}{\partial h}\frac{\partial h}{\partial \delta}}}_{\text{Non-differentiable manipulations and embedding!}}$$

<u>Solution</u>: Change The Optimizer



1. Compute gradient in feature space
2. Define a way for replacing values.
   For bytes: inverse look-up of embedding
3. Follow the direction of gradient and replace byte with other byte

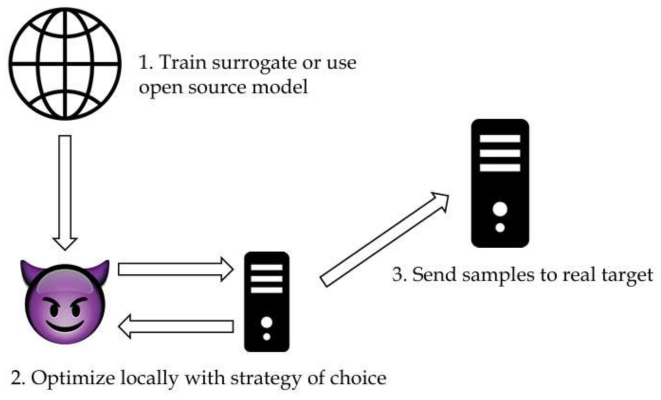
We can use **Gradient-free** (because Malware Detectors are not differentiable)

Reality check:

- *Robust models are not differentiable* (decision trees)
- *Most models are unavailable*: Most models are hosted on private servers, detection performed in cloud

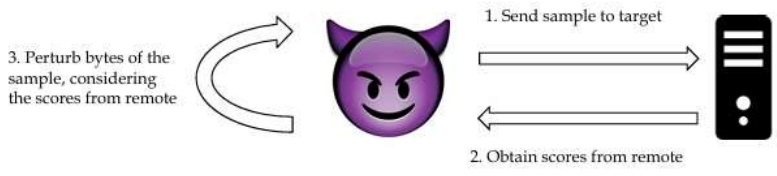
In these cases, **No gradients can be computed.**

1) Transfer Attacks


1. Train surrogate or use open source model
3. Send samples to real target
2. Optimize locally with strategy of choice

It has a problem: It is not training on the real model but on a **surrogate model.**

2) Query Attacks


3. Perturb bytes of the sample, considering the scores from remote
1. Send sample to target
2. Obtain scores from remote

I ask, I get an answer, If the answer is better than the previous one you keep that, you do some manipulations. (Then repeat the process). **Very slow if optimizer works byte-per-byte.**

3) Intead of modify byte by byte, I tell the optimizer to modify a big group of bytes, but not all the bytes, just the bytes that belong to the "**benign content".**
The optimizer explore less space, no modification byte-per-byte, but it relies on portions of goodware programs injected with practical manipulations.

PART 3.4

Physical Evasion Attacks: *EoT* and Adversarial Patches

You create adversarial example in the digital world. Your example is a vector which contains digital data x = […] → any single element can be manipulate as you want. But when you are in the real world, how can you manipulate, for example, a pencil? You want to fabricate a physical object which is recognized as another physical object.

The main concept behind the possibility of creating a *physical evasion attack* is related to the ***EoT* (Expectetion over Transformation).**

An Adversarial Example in Pysical World (not digital world): For example, **images** are **taken from** a **cell-phone camera.**

How Are They Optimized?

The constraint for the attacker here is to craft a perturbation which is **robust** to changes in *illumination, pose, distance to the camera*, etc.

I can realize it if I solve the problem called ***Expectetion over Transformation* (EoT).**
The physical object in the world can appear in many different poses and situation: all these variables are called **transformation** → so you have to take into account the **expectation** which represents all the possible transformations.

This name because because I have to do an **expectation** ($\mathbb{E}$). Usually it is a summary or an integral.

$$\begin{aligned} \underset{x'}{\arg\max} \quad & \mathbb{E}_{t\sim T}[\log P(y_t|t(x'))] \\ \text{subject to} \quad & \mathbb{E}_{t\sim T}[d(t(x'), t(x))] < \epsilon \\ & x \in [0,1]^d \end{aligned}$$

X' = adversarial example
X= original example (e.g. pencil)
E = expectation (it's a sum)
Y_t = lable of wrong object
t = transformation

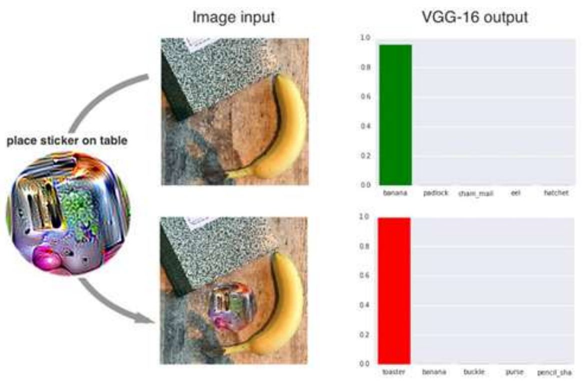
It is not enough that I create a perturbation able to evade the ML model, but I have to create a perturbation able to evade the ML model **for all possible transformation.** The constraint is that the perturbation has to be **robust** to changes in illumination, etc...
WE WANT TO MAXIMIZE THE PROBABILITY THAT AN OBJECT IS MISRECOGNIZED AS ANOTHER OBJECT.

So you are doing expectation E over all the possible transformation t. **T is the set of all the possible transformation**. T = [t1, t2, …]. **x** is the data you have.

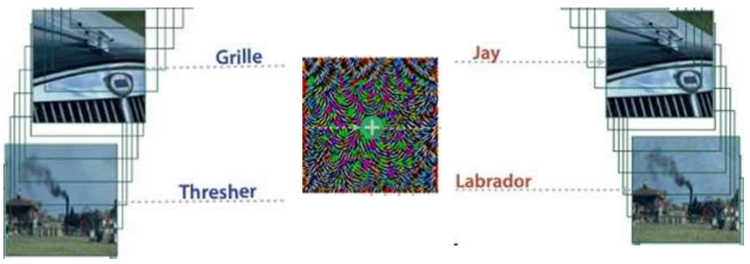
$\mathbb{E}_{t\sim T}$ With this mathematical expression we mean that we are considering all the possible transformation of the given data X. Also for the constraint, I have to do the Expectetion. The symbol **~T** means the Expectetion over all possible Transformation.

Adversarial Patch (Attack)

Instead of manipulating the physical object, I find a perturbation that can be added as a ***patch.***



We know that also exist ***Universal Adversarial Perturbations*** **(UAP),** which are perturbations (patches) that work for different object. So, They are called **UNIVERSAL** because the goal is to find **ONE SINGLE PERTURBATION** that is good for DIFFERENT REAL OBJECTS.



ADVERSARIAL EXAMPLE is different from UNIVERSAL ADVERSARIAL PERTURBATION (UAP)

ADVERSARIAL EXAMPLE = maximize the probability that an object is misrecognized as another object.

UNIVERSAL ADVERSARIAL PERTURBATION (UAP) = the goal is different: I want to find one single perturbation so that different objects are misrecognized. So I have multiple objects, and I create only one perturbation to misrecognized the object.

No gradients can be computed in these scenarios, BLACK-Box attacks are needed!

There are two cases when no gradients can be computed:

- We don't know the ML model we want to attack (black-box)
- I know f(x), but I can't compute the gradient, because f(x) is not differentiable.

All the evasion attacks are based on the assumption that I can compute the gradient, but sometimes the gradient is not computable. Because the *Loss Function* is not diffrentiable or maybe I do not know the *Loss Function*. So it is a **black-box attack** because I can **NOT** compute the **gradient.**

For black-box attacks there are different techinique:

1. Using ML model approximation
2. NOT using ML model approximation

The most simple technique usable is the **Black-box (Gradient free) *Transfer* Attacks.**

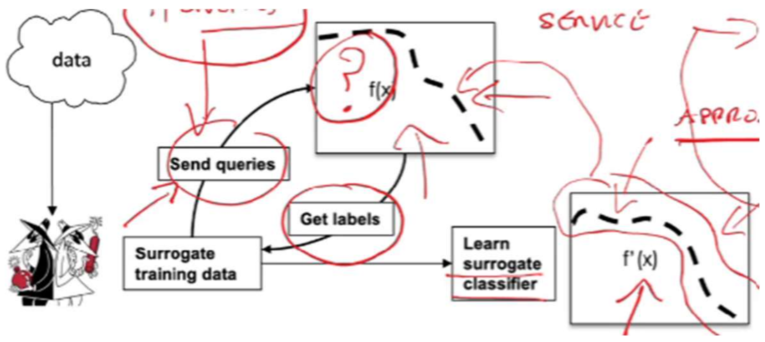
1) BLACK-BOX (GRADIENT FREE) *TRANSFER* ATTACKS

I should **create a surrogate model**, considering a close approximation of the real target model.

I select an approximation that is **differentiable**, so I can **compute the gradient for attack.**

At this point, we can **transfer the results:** I can use the Evasion Attack that we have seen, creating an adversarial perturbation and then I can try to attack the real ML model.

The **performance** depends on **how much the Surrogate Model is a good approximation.** So, people use many different models.



There are **limitations**:
1. Number of queries you need
2. Which data I use for queries
3. The model that I use for approximation.

The pratical issue is that I have to use a **SURROGATE MODEL** <u>that is a good approximation</u>.

Competition between models = training different surrogates and compute how they transfer "all vs all"

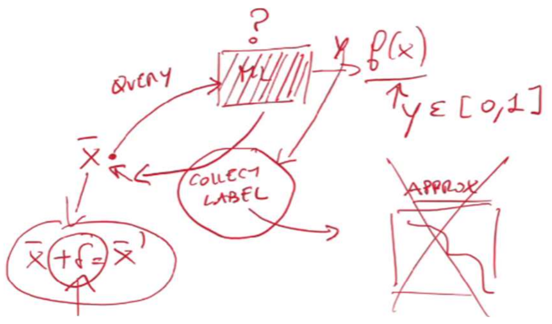
Different techniques have different results = the heatmap shows that different models behave differently when tested with attacks.

2) BLACK-BOX QUERY ATTACKS

There are some techniques which don't need to use the approximation, called **Black-box Query Attacks.** YOU DON'T NEED TO CREATE AN APPROXIMATION OF ML MODEL.

I take one initial example, I send the queries, I get just the recognition label, and depending on if the recognition label is right (so I'm able to evade) or I'm wrong, I try to generate a new example that is a little bit perturbated. USING THE LABEL, NOT AN APPROXIMATED MODEL.

Main challenge: keeping a small number of queries to stay undetected



COUNTERING EVASION ATTACKS: what can we do to **defend** our ML model

Defences means how can we make ML model **robust** against attacks.

There are techniques used for defence: **ADVERSARIAL (RE)TRAINING** is one of that. (it is done with Demetrio lab).

ADVERSARIAL TRAINING: Just training your model <u>also using adversarial example</u>.

(training set) + (adversarial examples)

It is **effective** because it is an experimental way for solving the real problem behind Adversarial Machine Learning

This is the mathematical problem behind the adversarial ML model. (l = loss function).

We want to **minimize** the Loss Function but we also want to take in account the **maximum** perturbation.

$$\min_{\boldsymbol{w}} \sum_i \max_{||\boldsymbol{\delta}_i||\leq\epsilon} \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i + \boldsymbol{\delta}_i))$$

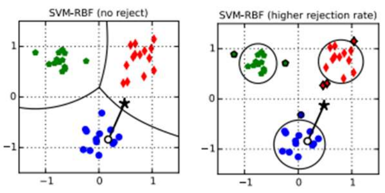
$\boxed{\textbf{bounded perturbation!}}$

This is called **ROBUST OPTIMIZATION**: is when we want to solve the minimum of the maximum

The only way is to use an adversarial example. Adversarial training is a good technique because it find the solution of the min-max problem.

So we:

1. Reduce sensitivity to input changes with robust optimization (adversarial training/regularization)
2. Introduce **rejection/detection** of adversarial examples



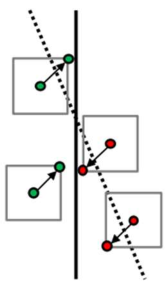
ROBUST OPTIMIZATION VIA ADVERSARIAL TRAINING (AT)

Why adversarial training is a solution of this min-max adversarial problem?

- Robust optimization (a.k.a. *adversarial training*)

$$\min_{\boldsymbol{w}} \max_{||\boldsymbol{\delta}_i||_\infty\leq\epsilon} \sum_i \ell(y_i, f_{\boldsymbol{w}}(\boldsymbol{x}_i + \boldsymbol{\delta}_i))$$

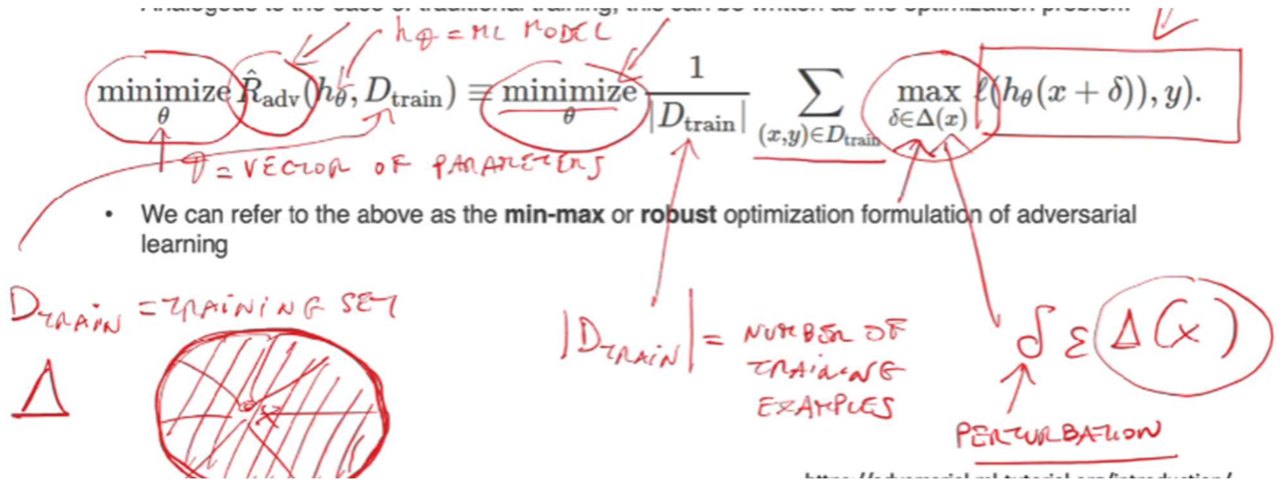
$\boxed{\textbf{bounded perturbation!}}$



To do a robust ML model, we need to solve this problem.

But if the loss function is NOT linear, solving this problem could be very difficult

The mathematical problem can be written in this way:



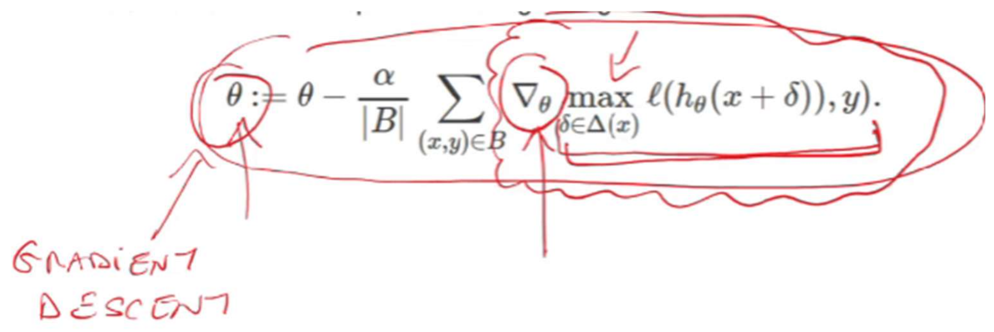
$$\underset{\theta}{\text{minimize}}\ \hat{R}_{\text{adv}}(h_\theta, D_{\text{train}}) \equiv \underset{\theta}{\text{minimize}}\ \frac{1}{|D_{\text{train}}|}\sum_{(x,y)\in D_{\text{train}}} \max_{\delta\in\Delta(x)} \boxed{\ell(h_\theta(x+\delta)), y).}$$

- We can refer to the above as the **min-max** or **robust** optimization formulation of adversarial learning

h_θ = ML MODEL
θ = VECTOR OF PARAMETERS
D_{TRAIN} = TRAINING SET
Δ
$|D_{TRAIN}|$ = NUMBER OF TRAINING EXAMPLES
$\delta \in \Delta(x)$
PERTURBATION

The min of the function is usually computed using the gradient. But the problem is the max in this case. I have to find the gradient of something which is the max of something else. But there is the solution to this difficult problem

You can use gradient descent as following:



$$\theta := \theta - \frac{\alpha}{|B|}\sum_{(x,y)\in B} \nabla_\theta \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y).$$

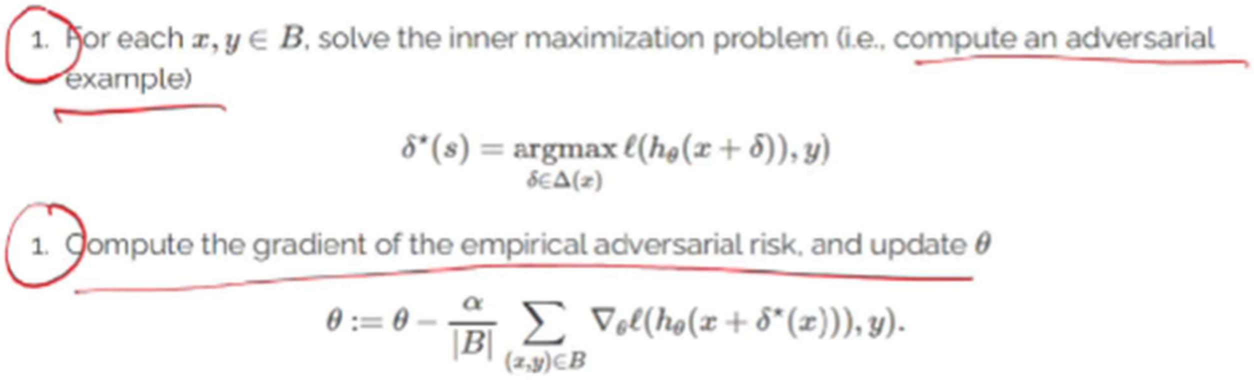
To do that I can use the **Danskin’s theorem**: I can compute the gradient of the maximum as the gradient of the loss function for the point that takes the maximum value.

So I compute the point delta* that provide the max of the loss function as following:

$$\delta^\star = \underset{\delta\in\Delta(x)}{\text{argmax}}\ \boxed{\ell(h_\theta(x+\delta)), y)}$$

Then you can compute the gradient for the point where you have the max point:

$$\boxed{\nabla_\theta \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y)} = \boxed{\nabla_\theta \ell(h_\theta(x+\delta^\star)), y)}$$



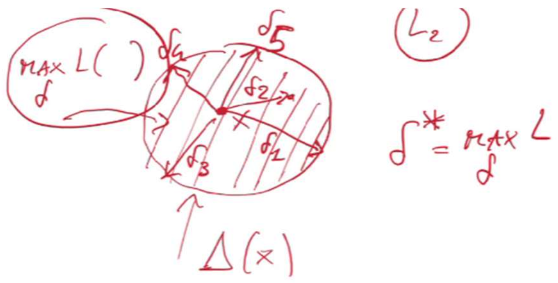
1. For each $x, y \in B$, solve the inner maximization problem (i.e., compute an adversarial example)

$$\delta^*(s) = \operatorname*{argmax}_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y)$$

1. Compute the gradient of the empirical adversarial risk, and update θ

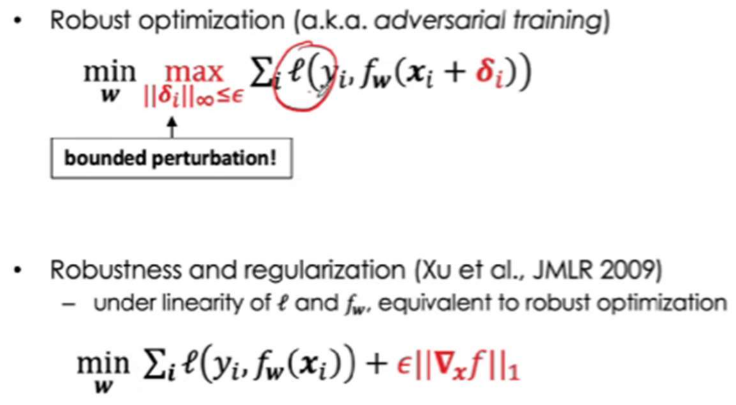
$$\theta := \theta - \frac{\alpha}{|B|} \sum_{(x,y)\in B} \nabla_\theta \ell(h_\theta(x+\delta^*(x))), y).$$

So adversarial training is only compute the attack and then retraining. So we train the model using the adversarial example, too.



ROBUST OPTIMIZATION VIA REGULARIZATION

Only when **the loss function is linear**, the complicated min-max problem can be writter as a minimization problem, but we have to add a **minimization term.**



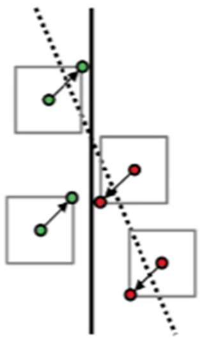
- Robust optimization (a.k.a. *adversarial training*)

$$\min_{w} \max_{||\delta_i||_\infty \leq \epsilon} \Sigma_i \ell(y_i, f_w(x_i + \delta_i))$$

bounded perturbation!

- Robustness and regularization (Xu et al., JMLR 2009)
  - under linearity of ℓ and f_w, equivalent to robust optimization

$$\min_{w} \Sigma_i \ell(y_i, f_w(x_i)) + \epsilon||\nabla_x f||_1$$



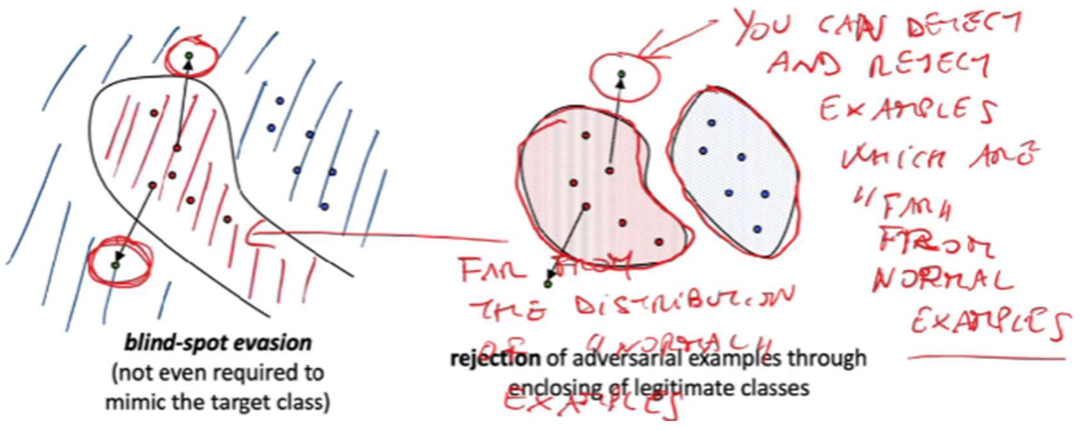
For the maximization problem, the norm is the infinite norm and in the regularization term we have the gradient of the function, but the norm is the "unitary" norm.

When you want to transform the **min-max** problem in a **minimization + regularization term** problem, there is a sort of DUALITY: if one norm is "infinite", the other norm should be "unitary" (and viceversa).

Adversarial training is the most used technique to make a ML model more robust, but there are also other techniques:

COUNTERING EVASION: DETECTING & REJECTION ADVERSARIAL EXAMPLES

If Adversarial example are quite far from normal example, another simple defence mechanism that you can use is trying to create a ML model which has the circular boundary close surrounding the distribution of normal examples.



YOU CAN DETECT AND REJECT EXAMPLES WHICH ARE "FAR" FROM NORMAL EXAMPLES
FAR FROM THE DISTRIBUTION OF NORMAL EXAMPLES
blind-spot evasion
(not even required to mimic the target class)
rejection of adversarial examples through enclosing of legitimate classes

So you can detect and reject examples which are “far” from normal examples.

Possible method: using DNN (Deep Neural Network)