

Adversarial Machine Learning

Machine Learning Course - CS-433

Nov 13, 2024

Nicolas Flammarion

EPFL

Some input examples are hard for humans



Dog or mop?

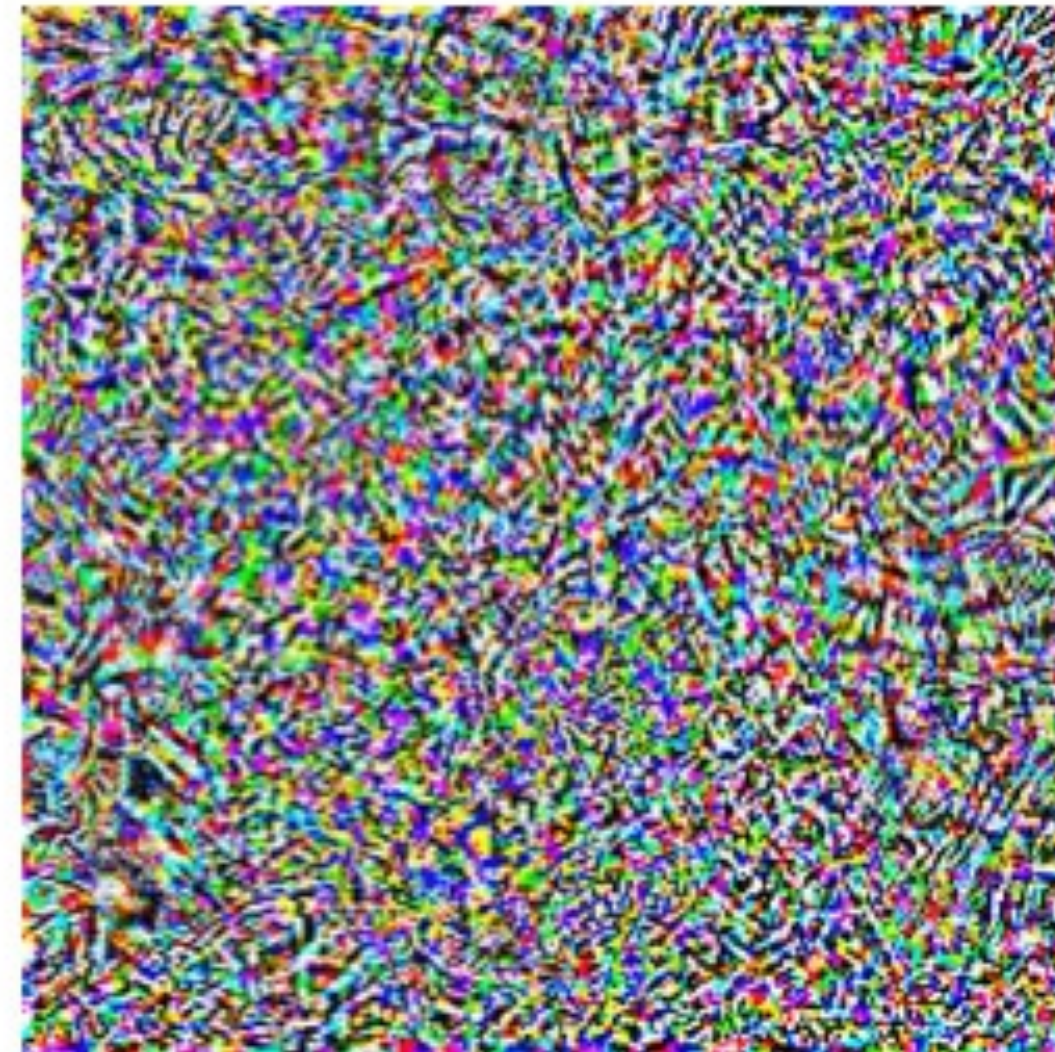
- Some examples may be challenging for humans
- NNs typically have **no problem** with them
- However, NNs are not always **robust** in their decisions

Adversarial examples are small perturbations that cause misclassification with high confidence

“pig”



+ 0.005 x



=



“airliner”

Source: Z. Kolter, A. Madry, NeurIPS'18 tutorial on adversarial robustness

NNs have struggle with imperceptible yet very specific inputs known as **adversarial examples**

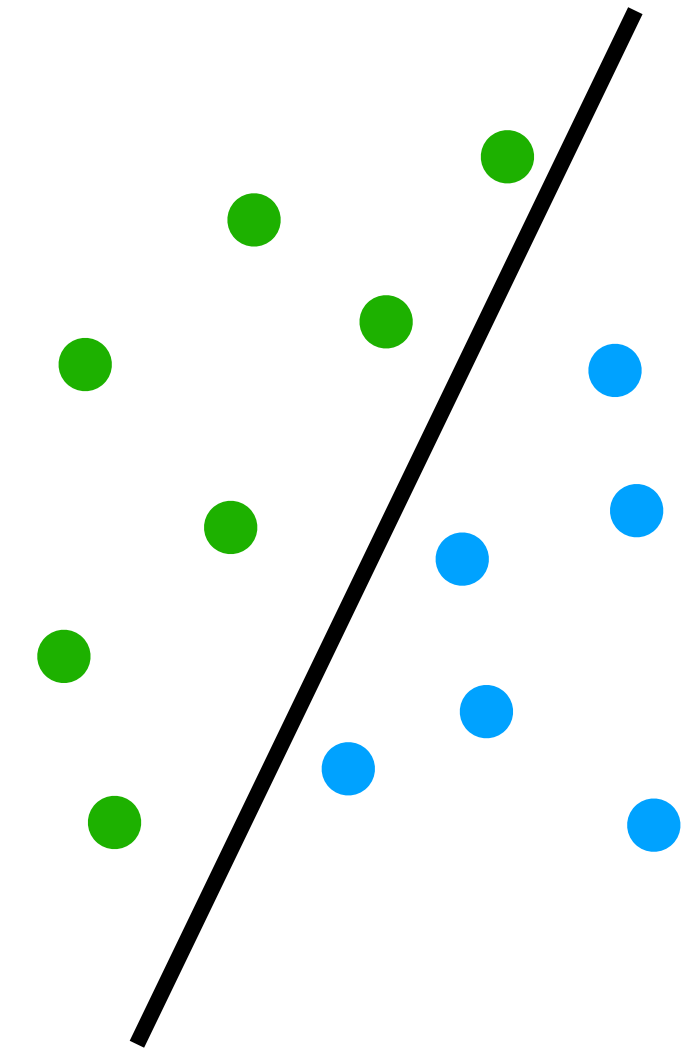
- ➡ **Security issue:** consider the implications for a self-driving car and its ability to detect stop signs
- ➡ We don't understand how these models **generalize** and react to shifts in the distribution of data (i.e., **distribution shifts**)

Standard risk

Classification problem: $(X, Y) \sim \mathcal{D}$, Y with range $\{-1, 1\}$

Standard risk: average zero-one loss over X

$$R(f) = \mathbb{E}_{\mathcal{D}} \left[1_{f(X) \neq Y} \right] = \mathbb{P}_{\mathcal{D}} \left[f(X) \neq Y \right]$$



Standard risk vs. adversarial risk

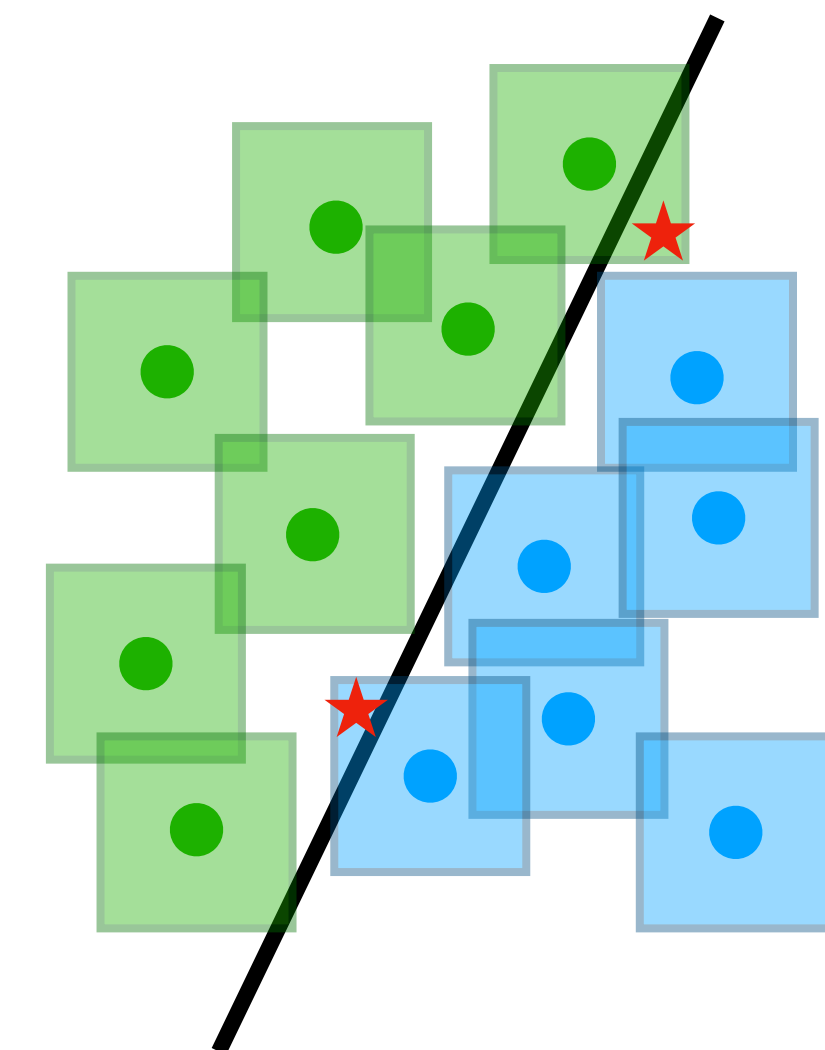
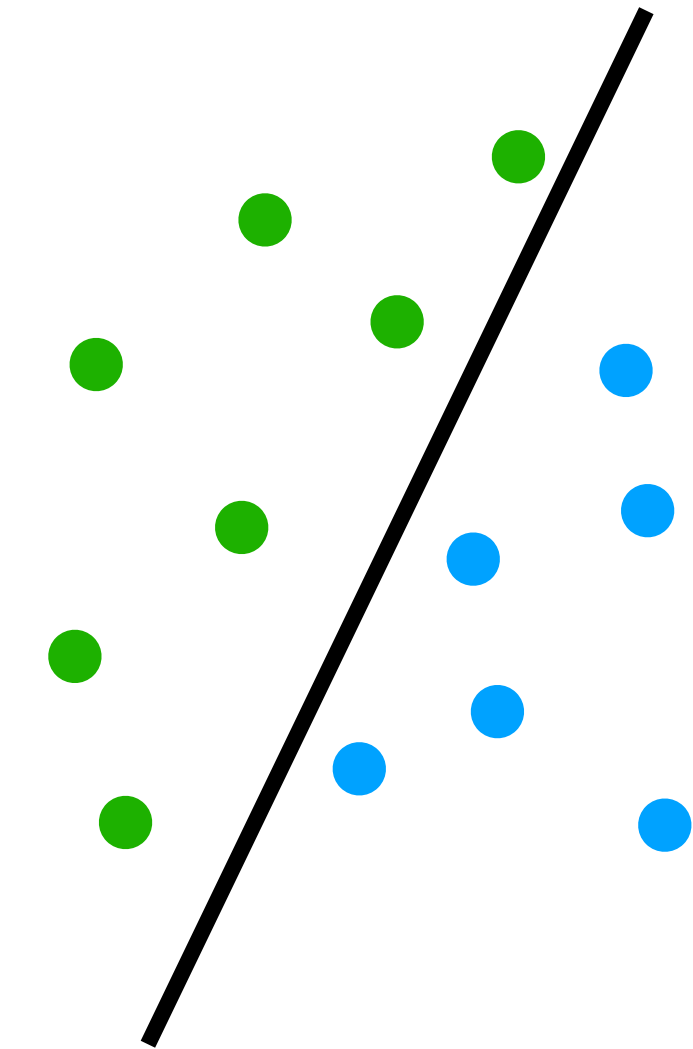
Classification problem: $(X, Y) \sim \mathcal{D}$, Y with range $\{-1, 1\}$

Standard risk: average zero-one loss over X

$$R(f) = \mathbb{E}_{\mathcal{D}} \left[1_{f(X) \neq Y} \right] = \mathbb{P}_{\mathcal{D}} [f(X) \neq Y]$$

Adversarial risk: average zero-one loss over **small, worst-case perturbations of X**

$$R_{\varepsilon}(f) = \mathbb{E}_{\mathcal{D}} \left[\max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \right]$$



Adversarial vulnerability raises many questions

$$R_\varepsilon(f) = \mathbb{E}_{\mathcal{D}} \left[\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \right]$$

- Threat model:
 - How should we define the adversary's power?
 - Which norm should we consider? ℓ_∞ , ℓ_2 , ℓ_1 , ℓ_0 , ...
 - What set of perturbations?
- If $R(f) \leq \delta$, then how large can $R_\varepsilon(f)$ be?

Adversarial vulnerability raises many questions

- How can we compute an adversarial example?
- What level of access do we have to the model to attack it?
- How can we design a classifier f so that it is robust?
Related: given a non-robust classifier, how can we make it robust?
- Why are neural networks non-robust?

Generating adversarial examples

Task: given an input (x, y) and a model

$f: \mathcal{X} \rightarrow \{-1, 1\}$, find an input \hat{x} , such that

(a) $\|\hat{x} - x\| \leq \varepsilon$

(b) the model f makes a mistake on it

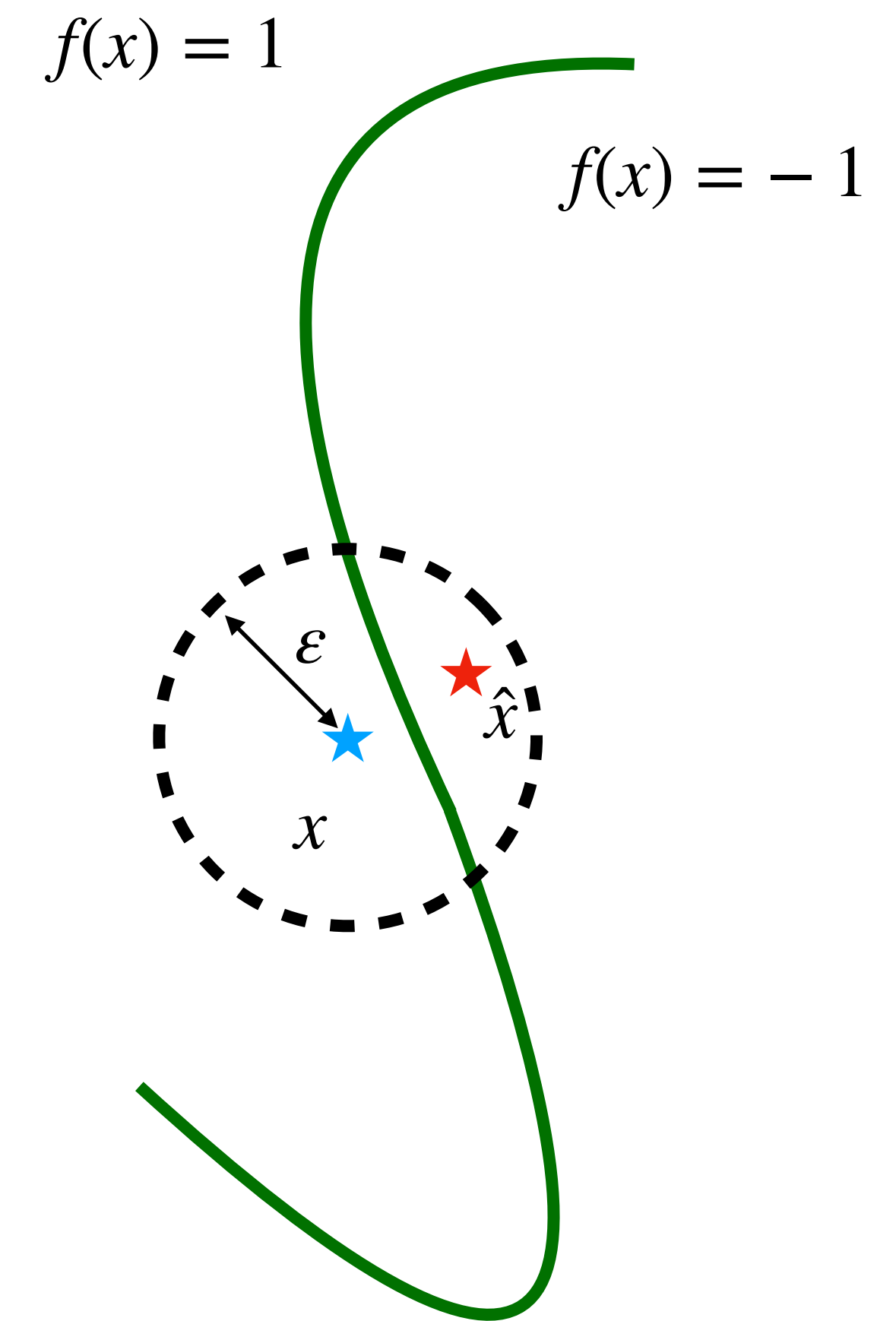
Trivial case: x is already misclassified

➡ No action required

General case: x is correctly classified

➡ find \hat{x} such that $f(\hat{x}) \neq y$ and $\|\hat{x} - x\| \leq \varepsilon$

i.e., $\hat{x} \in B_x(\varepsilon) \cap \{x', f(x') = -y\}$



Generating adversarial examples amounts to maximizing the classification loss w.r.t. the inputs

Find an adversarial example by solving

$$\max_{\hat{x}, \|\hat{x}-x\|\leq\epsilon} 1_{f(\hat{x})\neq y}$$

➡ Optimization problem with respect to the inputs

Problem: optimizing the indicator function $1_{f(\hat{x})\neq y}$ is difficult:

1. The indicator function 1 is not continuous
2. The NN prediction f outputs discrete class values $\{-1, 1\}$

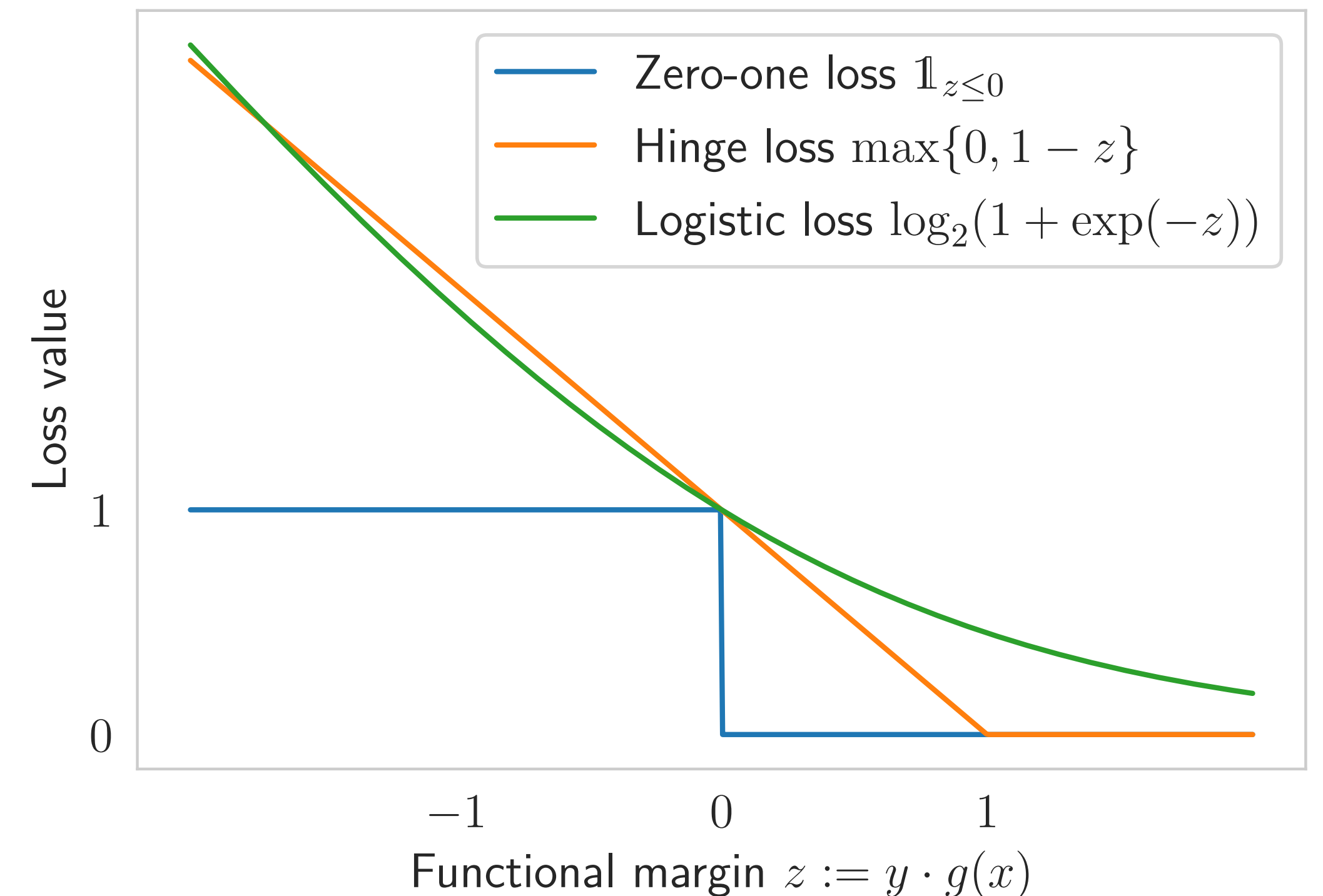
Generating adversarial examples amounts to solving a constrained optimization problem

Solution:

1. Use a smooth classification loss ℓ (e.g., logistic or hinge loss) instead
2. Consider the output g of the NN before classification (i.e., $f(x) = \text{sign}(g(x))$)

Main idea: Replace the difficult problem involving the indicator with a smooth problem

$$\max_{\hat{x}, \|\hat{x}-x\| \leq \varepsilon} \mathbb{1}_{f(\hat{x}) \neq y} \longrightarrow \max_{\hat{x}, \|\hat{x}-x\| \leq \varepsilon} \ell(yg(\hat{x}))$$



Reminder: decreasing, margin-based (i.e., dependent on $y \cdot g(x)$) classification losses

Generating adversarial examples: white-box case


How to solve $\max_{\hat{x}, \|\hat{x}-x\| \leq \epsilon} \ell(yg(\hat{x}))$ in the **white-box** case, i.e., if we know the model g ?

Compute its gradient: $\nabla_x \ell(yg(x)) = \underbrace{y \ell'(yg(x))}_{\leq 0 \text{ since classification loss are decreasing}} \nabla_x g(x)$

We should move in the direction $\propto -y \nabla_x g(x)$

Interpretation: $f(x) = \text{sign}(g(x))$

- If $y = 1$, we want to decrease $g(x)$ and follow $-\nabla_x g(x)$
- If $y = -1$, we want to increase $g(x)$ and follow $\nabla_x g(x)$

 Why use ℓ , and not directly minimize $yg(\hat{x})$?
→ It won't extend to multi-class classification and robust training.

Generating adversarial examples: taking into account the constraints

We can linearize the loss $\tilde{\ell}(x) := \ell(yg(x))$ to derive an iteration:

$$\begin{aligned}\max_{\|\hat{x}-x\|\leq\epsilon} \tilde{\ell}(\hat{x}) &\approx \max_{\|\hat{x}-x\|\leq\epsilon} \tilde{\ell}(x) + \nabla_x \tilde{\ell}(x)^T (\hat{x} - x) \\ &= \tilde{\ell}(x) + \max_{\|\hat{x}-x\|\leq\epsilon} \nabla_x \tilde{\ell}(x)^T (\hat{x} - x) \\ &= \tilde{\ell}(x) + \max_{\|\delta\|\leq\epsilon} \nabla_x \tilde{\ell}(x)^T \delta\end{aligned}$$

- We need to maximize the inner product under a norm constraint, i.e. find the optimal local update
- This is a simple problem for which we can get a closed-form solution depending on the norm used to measure the perturbation size $\|\delta\|$

Generating adversarial examples: one-step attack

Problem:

$$\max_{\|\delta\| \leq \varepsilon} \nabla_x \tilde{\ell}(x)^T \delta$$

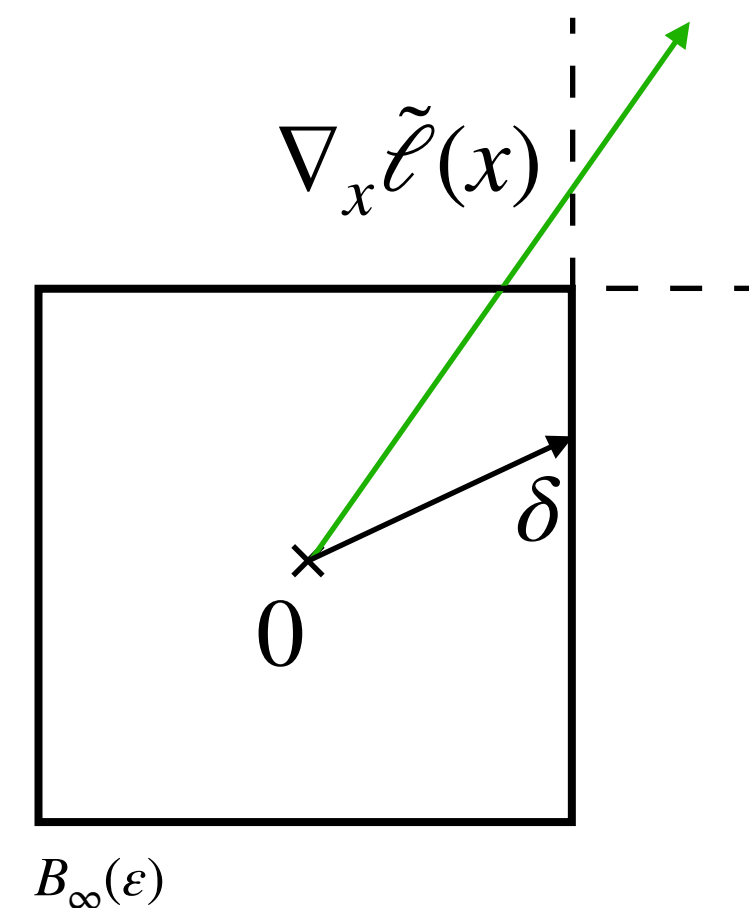
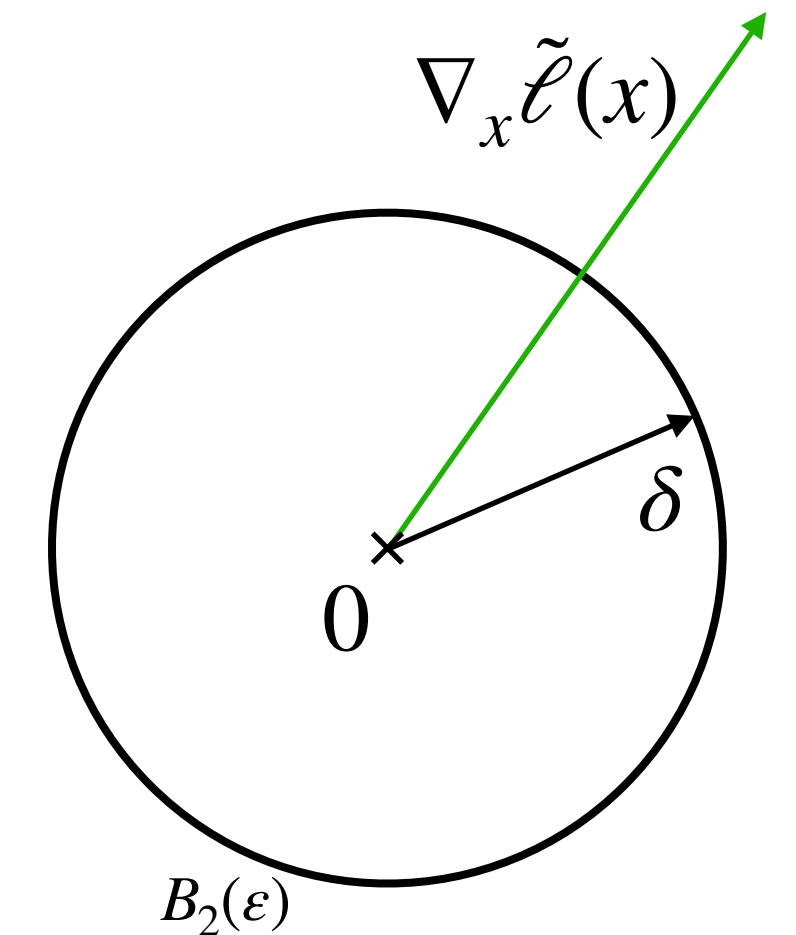
- Solution for the ℓ_2 norm: $\delta_2^* = \varepsilon \cdot \frac{\nabla_x \tilde{\ell}(x)}{\|\nabla_x \tilde{\ell}(x)\|_2} = -\varepsilon y \cdot \frac{\nabla_x g(x)}{\|\nabla_x g(x)\|_2}$

$$\Rightarrow \hat{x} = x - \varepsilon y \cdot \frac{\nabla_x g(x)}{\|\nabla_x g(x)\|_2}$$

- Solution for the ℓ_∞ norm: $\delta_\infty^* = \varepsilon \cdot \text{sign}(\nabla_x \tilde{\ell}(x)) = -\varepsilon y \cdot \text{sign}(\nabla_x g(x))$

$$\Rightarrow \hat{x} = x - \varepsilon y \cdot \text{sign}(\nabla_x g(x))$$

Fast Gradient Sign Method
[Goodfellow et al., 2014]



Generating adversarial examples: multi-step attack

These updates can be done iteratively and combined with a projection Π on the feasible set (i.e., ℓ_2/ℓ_∞ balls here)

Projected Gradient Descent (PGD attack):

- ℓ_2 norm:

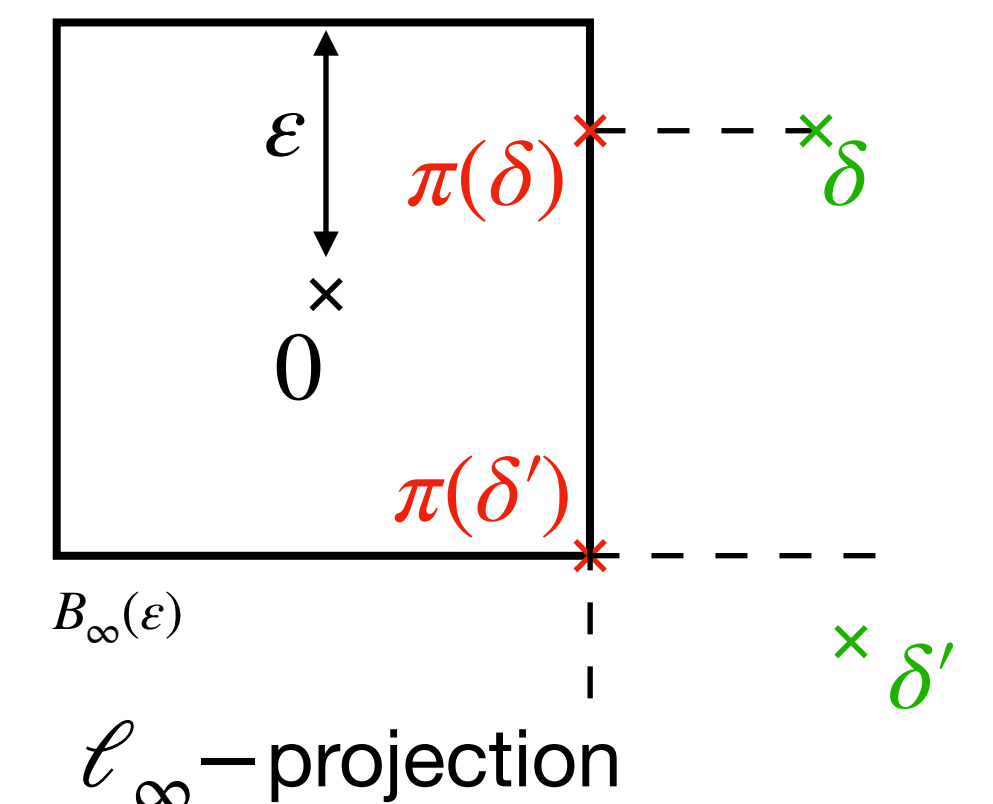
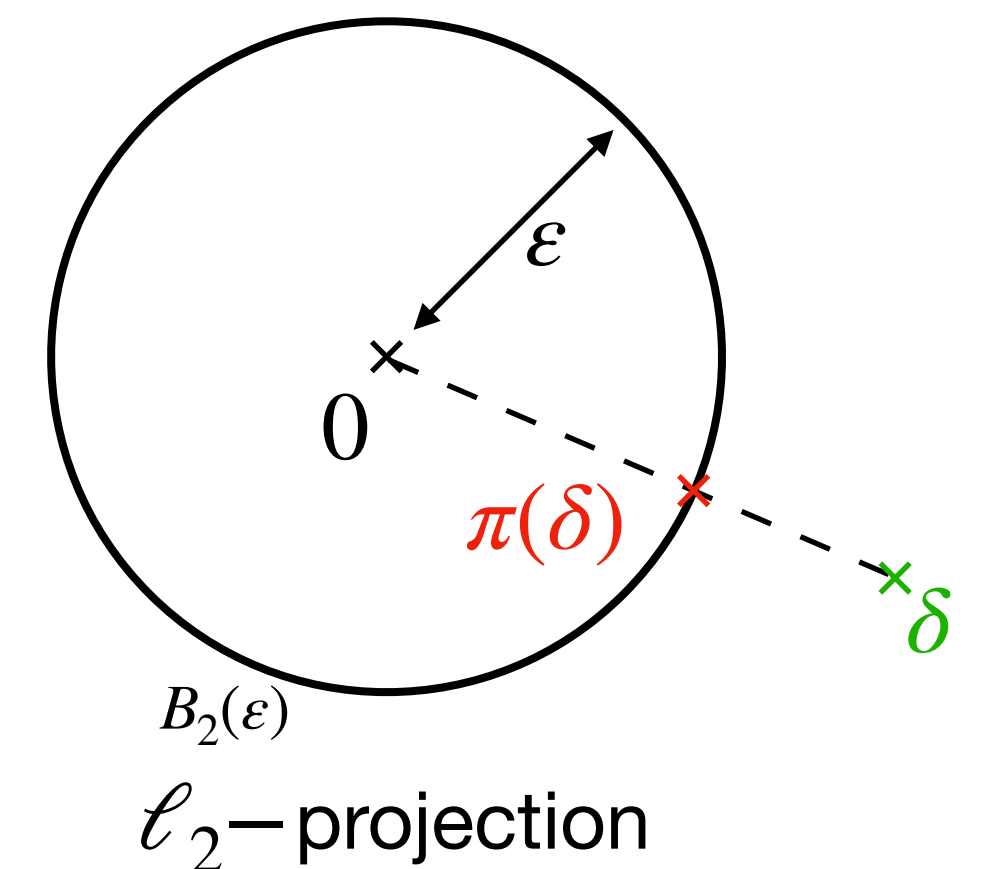
$$\delta^{t+1} = \Pi_{B_2(\epsilon)} \left[\delta^t + \alpha \cdot \frac{\nabla \tilde{\ell}(x + \delta^t)}{\|\nabla \tilde{\ell}(x + \delta^t)\|_2} \right],$$

$$\text{where } \Pi_{B_2(\epsilon)}(\delta) = \begin{cases} \epsilon \cdot \delta / \|\delta\|_2, & \text{if } \|\delta\|_2 \geq \epsilon \\ \delta, & \text{otherwise} \end{cases}$$

- ℓ_∞ norm:

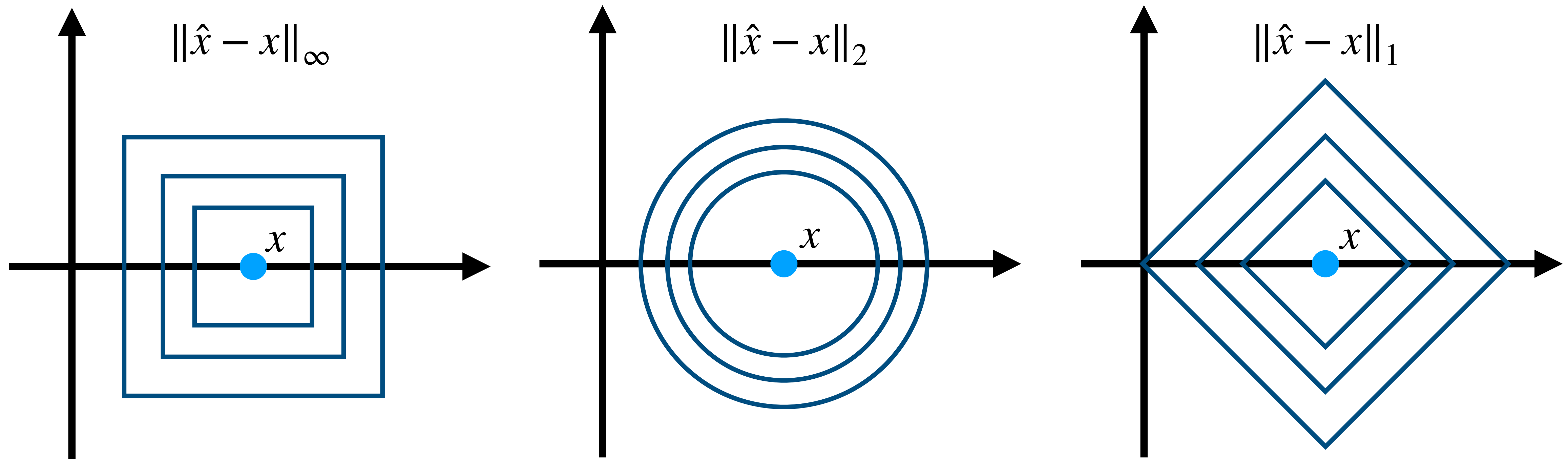
$$\delta^{t+1} = \Pi_{B_\infty(\epsilon)} \left[\delta^t + \alpha \cdot \text{sign}(\nabla \tilde{\ell}(x + \delta^t)) \right],$$

$$\text{where } \Pi_{B_\infty(\epsilon)}(\delta)_i = \begin{cases} \epsilon \cdot \text{sign}(\delta_i), & \text{if } |\delta_i| \geq \epsilon \\ \delta_i, & \text{otherwise} \end{cases}$$



Reminder: ℓ_p norms

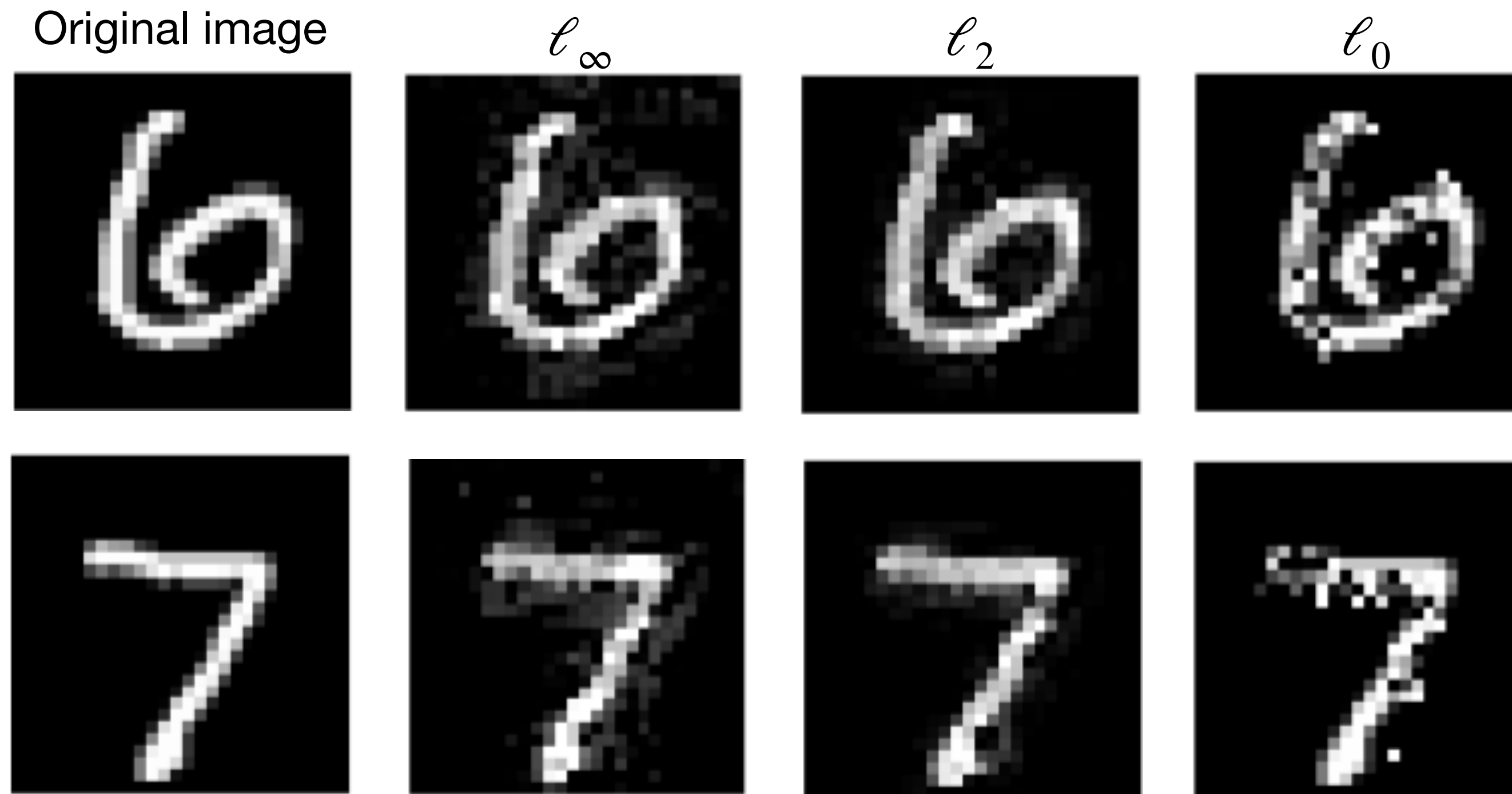
Different ℓ_p norms have different geometry



The difference is especially pronounced in high dimensions!

Visualizations of different ℓ_p adversarial examples

The choice of the norm leads to different properties of the resulting adversarial perturbations: e.g. ℓ_∞ are **dense** and ℓ_0 are **sparse**



Source: Towards Evaluating the Robustness of Neural Networks, Carlini et al., 2018

Which perturbations should we aim to be robust against?

➡ Extensive research on formulating the 'right' perturbation set!

White-box attacks: implementation

- For a neural network, the gradients $\nabla_x g(x)$ can also be computed by **backpropagation** (note: they are taken w.r.t. **inputs**, not parameters!)
- Modern deep learning frameworks readily support this
→ **lab #9** (implement Fast Gradient Sign Method on MNIST in PyTorch)
- Now: what **if we don't know** $g(x)$? Is it possible to run an attack without knowing how to compute the gradient $\nabla_x g(x)$?

Black-box attacks: query-based gradient estimation

There are different assumptions on the knowledge about the model f :

- **score-based:** we can query the continuous model scores $g(x) \in \mathbb{R}$
- **decision-based:** we can query only the predicted class $f(x) \in \{-1, 1\}$

In the score-based case, we can approximate the gradient by using the finite difference formula:

$$\nabla_x g(x) \approx \sum_{i=1}^d \frac{g(x + \alpha e_i) - g(x)}{\alpha} e_i$$

Remark: similar techniques can be adapted for the decision-based case (when x is near the decision boundary)

Black-box attacks via transfer attacks

Alternative approach: **transfer attacks**

1. train a **similar** surrogate model $\hat{f} \approx f$ on **similar** data
 2. transfer the resulting white-box adversarial perturbation from \hat{f} to f
- Success depends on how **similar** the model architecture and data are
 - If we are allowed to query f given some **unlabeled** inputs $\{x_n\}_{n=1}^N$, we can obtain $\{x_n, f(x_n)\}_{n=1}^N$ and use that information to learn \hat{f} (known as **model stealing**)
 - can facilitate **transfer attacks**

Black-box attacks: summary

General takeaway: black-box attacks are of practical concern but:

- Query-based methods often require a lot of queries (10k-100k), particularly **decision-based attacks** → easy to restrict access for the attacker!
- Obtaining a surrogate model \hat{f} can be costly and there is no guarantee of success
- A critical missing element is the implementation of **physically realizable attacks**

Physically realizable attacks

For practical application, adversarial examples must meet additional requirements:

- Resilience to JPEG compression (for images input directly in a digital format)
- Resilience to photographic distortions (for real-world adversarial examples captured by a camera)
- Resilience to varying camera angles (for a moving camera, e.g., on a self-driving car)

→ a surge of papers on how to take these requirements into account



Source: Robust Physical-World Attacks on Deep Learning Visual Classification (CVPR 2018)

How do we train robust models?

We have seen how to **generate** adversarial examples, but **how can we make our models robust to such attacks?**

➡ Simply train them on these adversarial examples, a.k.a. **adversarial training**

- **Standard training:** the goal is to minimize the **standard risk**:

$$\min_{\theta} R(f_{\theta}) = \mathbb{E}_{\mathcal{D}} \left[1_{f(X) \neq Y} \right]$$

- **Adversarial training:** the goal is to minimize the **adversarial risk**:

$$\min_{\theta} R_{\varepsilon}(f_{\theta}) = \mathbb{E}_{\mathcal{D}} \left[\max_{\hat{x}, \|\hat{x} - X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \right]$$

Adversarial training: formulation

Goal:

$$\min_{\theta} R_{\varepsilon}(f_{\theta}) = \mathbb{E}_{\mathcal{D}} \left[\max_{\hat{x}, \|\hat{x}-X\| \leq \varepsilon} 1_{f(\hat{x}) \neq Y} \right]$$

- The data distribution \mathcal{D} is unknown \rightarrow approximate it with a **sample average**
- The classification loss is non-continuous \rightarrow use a **smooth loss**

This leads to the following **robust optimization** problem:


$$\min_{\theta} \frac{1}{N} \sum_{n=1}^N \max_{\hat{x}_n, \|x_n - \hat{x}_n\| \leq \varepsilon} \ell(y_n g_{\theta}(\hat{x}_n))$$

Interpretation: minimize the risk on adversarial examples

Adversarial training: algorithm

$$\min_{\theta} \frac{1}{N} \sum_{n=1}^N \max_{\hat{x}_n, \|x_n - \hat{x}_n\| \leq \varepsilon} \ell(y_n g_{\theta}(\hat{x}_n))$$

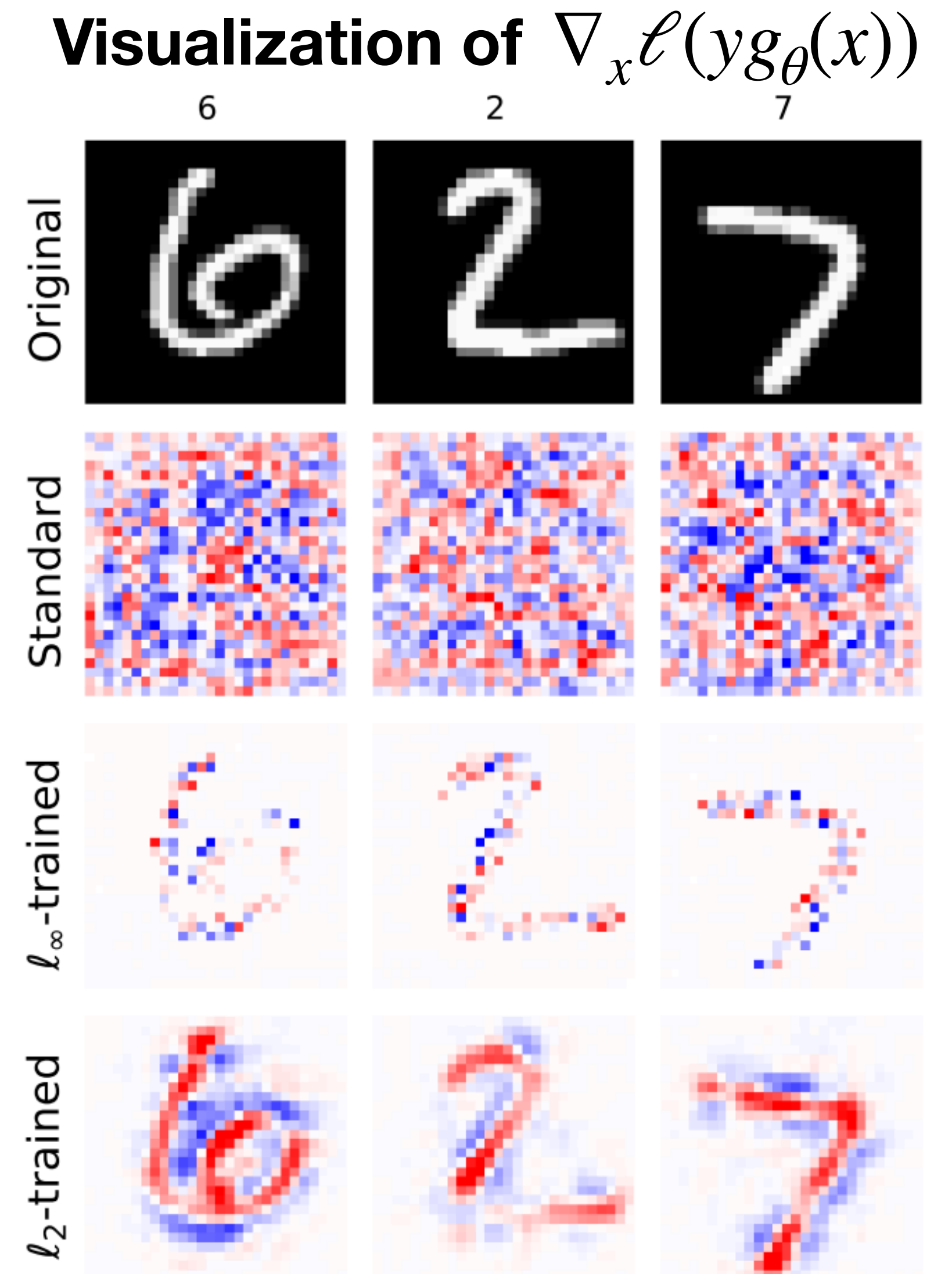
Adversarial training: at each iteration t :

1. For each x_n , approximate $\hat{x}_n^{\star} \approx \arg \max_{\|x_n - \hat{x}_n\| \leq \varepsilon} \ell(y_n g_{\theta}(\hat{x}_n))$ via the **PGD attack**
2. Perform a gradient descent step w.r.t. θ using $\frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \ell(y_n g_{\theta}(\hat{x}_n^{\star}))$
 Note you are using \hat{x}_n^{\star} and not x_n

Adversarial training: discussion

Good news:

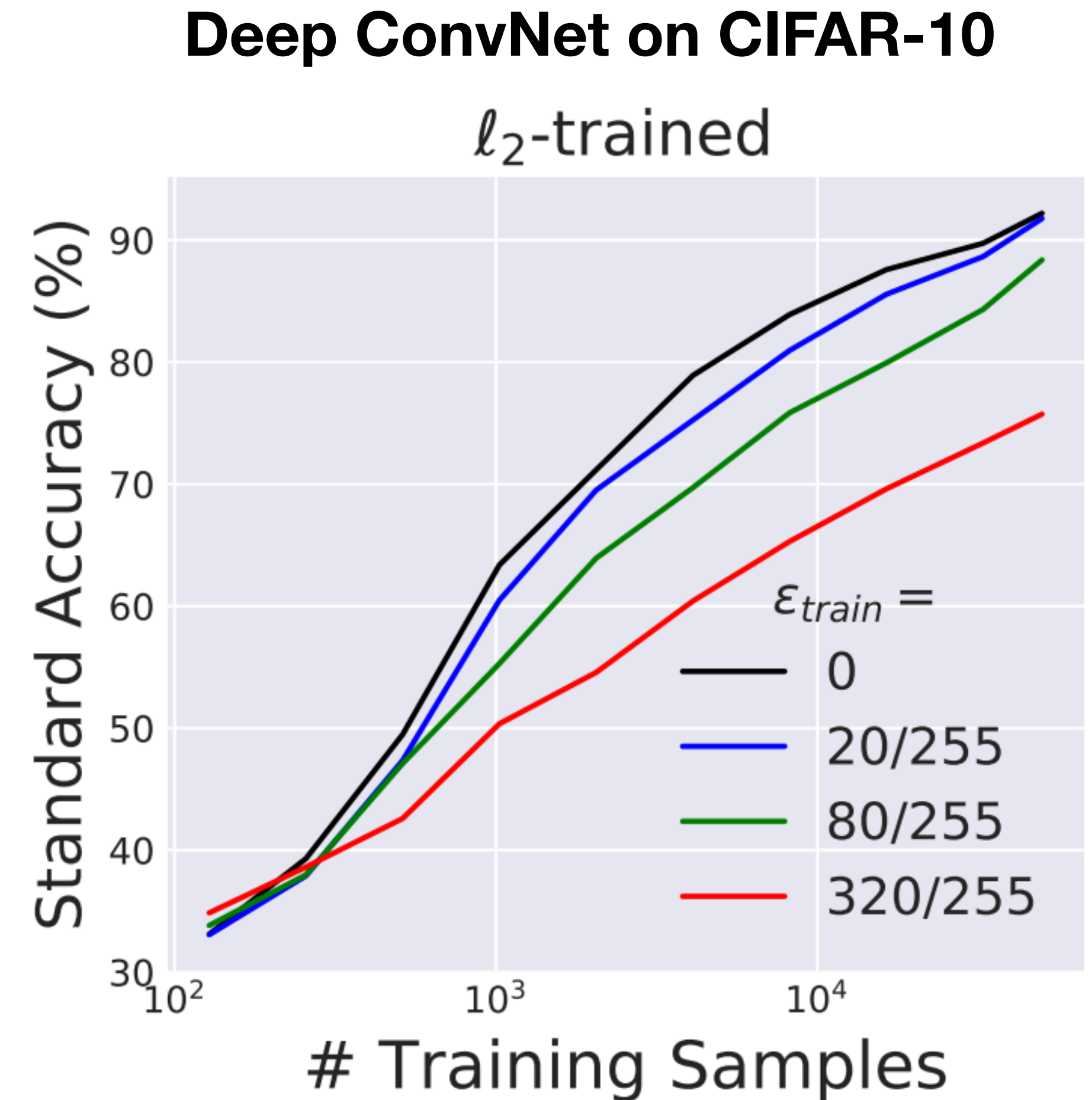
- Adversarial training is a state-of-the-art approach for robust classification!
- Adversarial training leads to **more interpretable gradients** $\nabla_x \ell(yg_\theta(x))$
- The algorithm is fully compatible with SGD
→ you will explore it in **lab #9**
(adversarial training of a CNN on MNIST)



Adversarial training: discussion

Bad news:

- Increased computational time: proportional to the number of PGD steps
- **Robustness-accuracy tradeoff:** using too large ϵ lead to worse standard accuracy (right)



Source: Robustness May Be at Odds with Accuracy (ICLR 2019)

Key question: so why do adversarial examples exist?

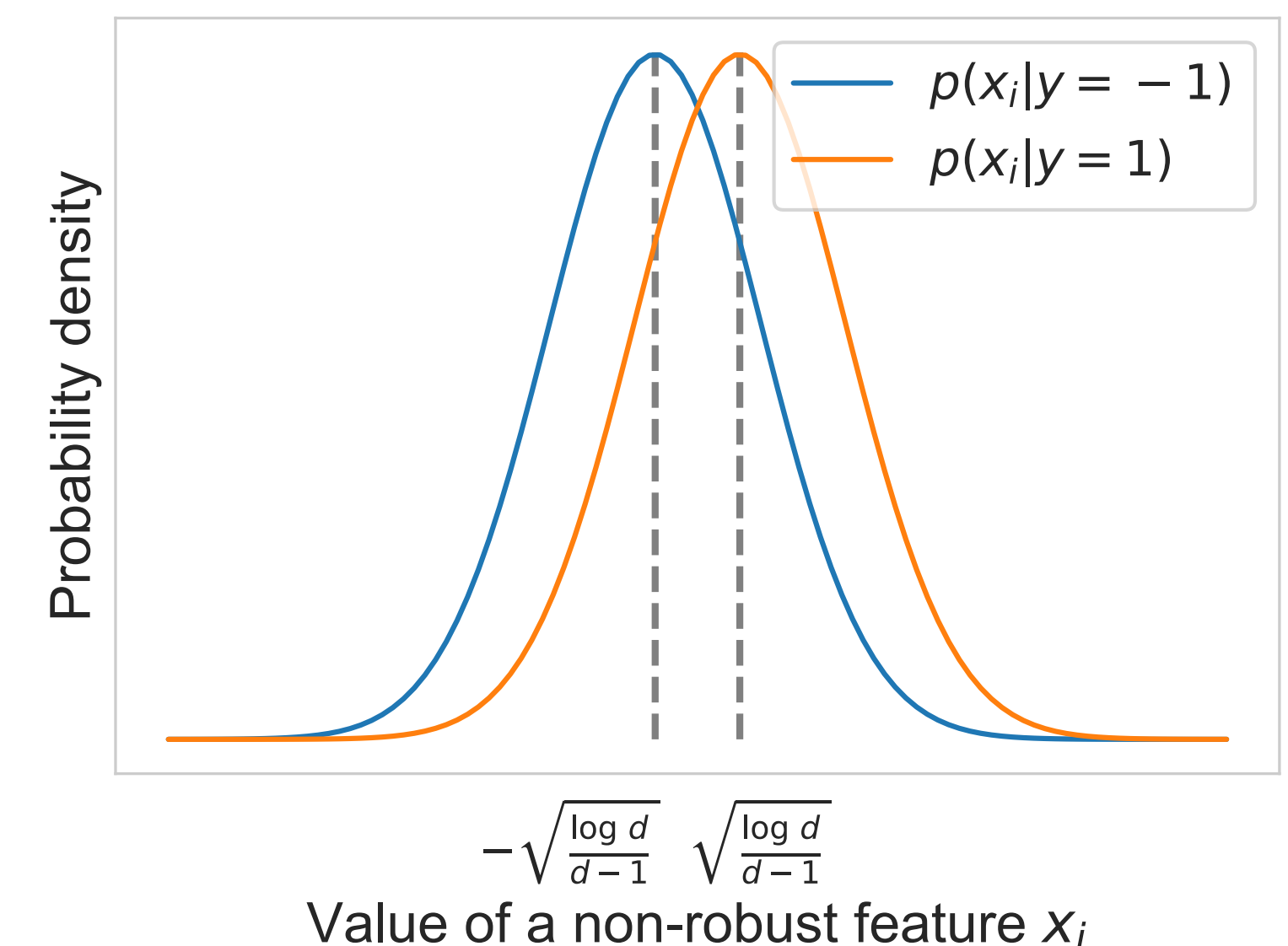
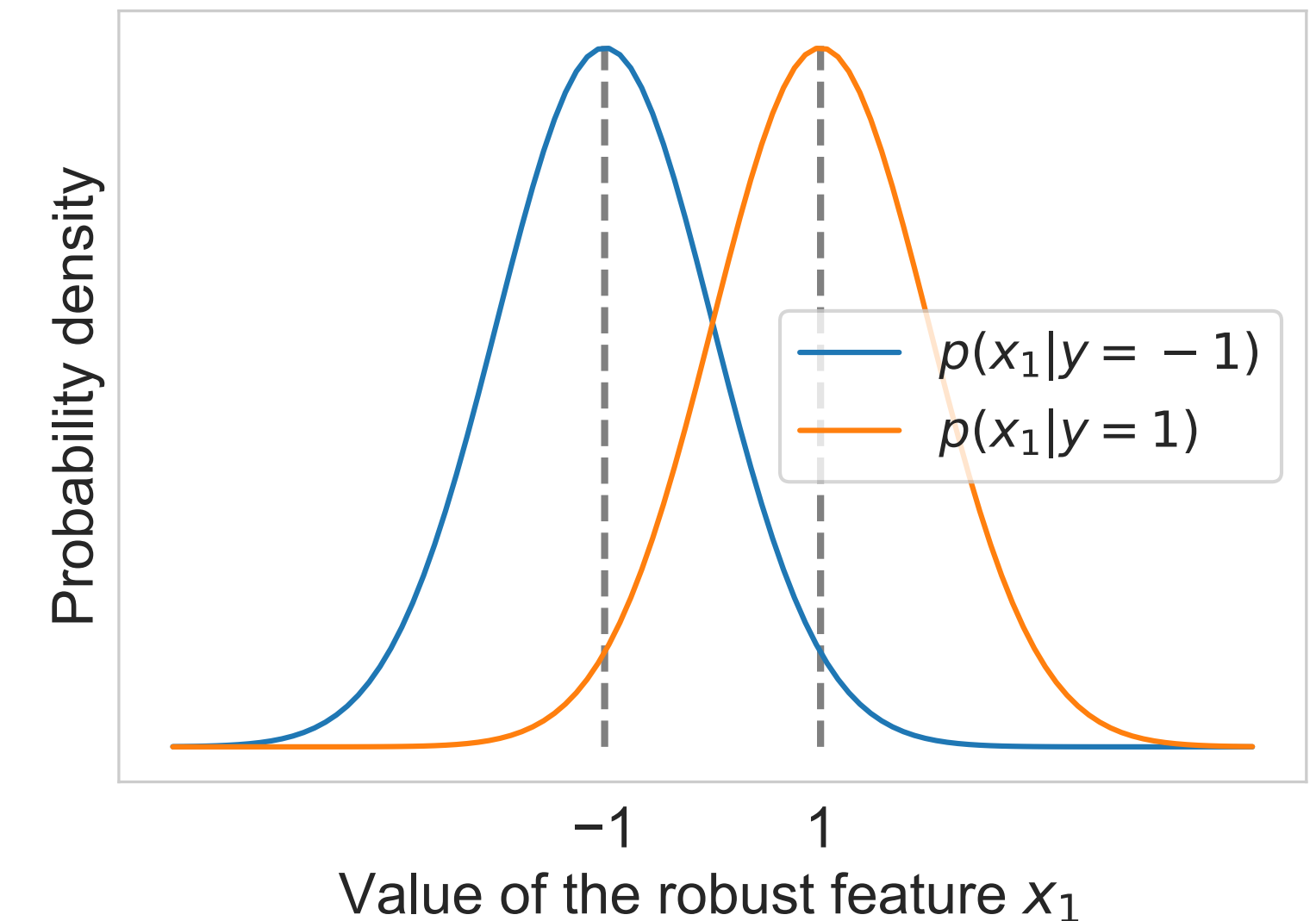
We can conceptualize it with a simple model

Consider $x \in \mathbb{R}^d$, $y \sim \mathcal{U}(\{-1, 1\})$, $Z_i \sim \mathcal{N}(0, 1)$:

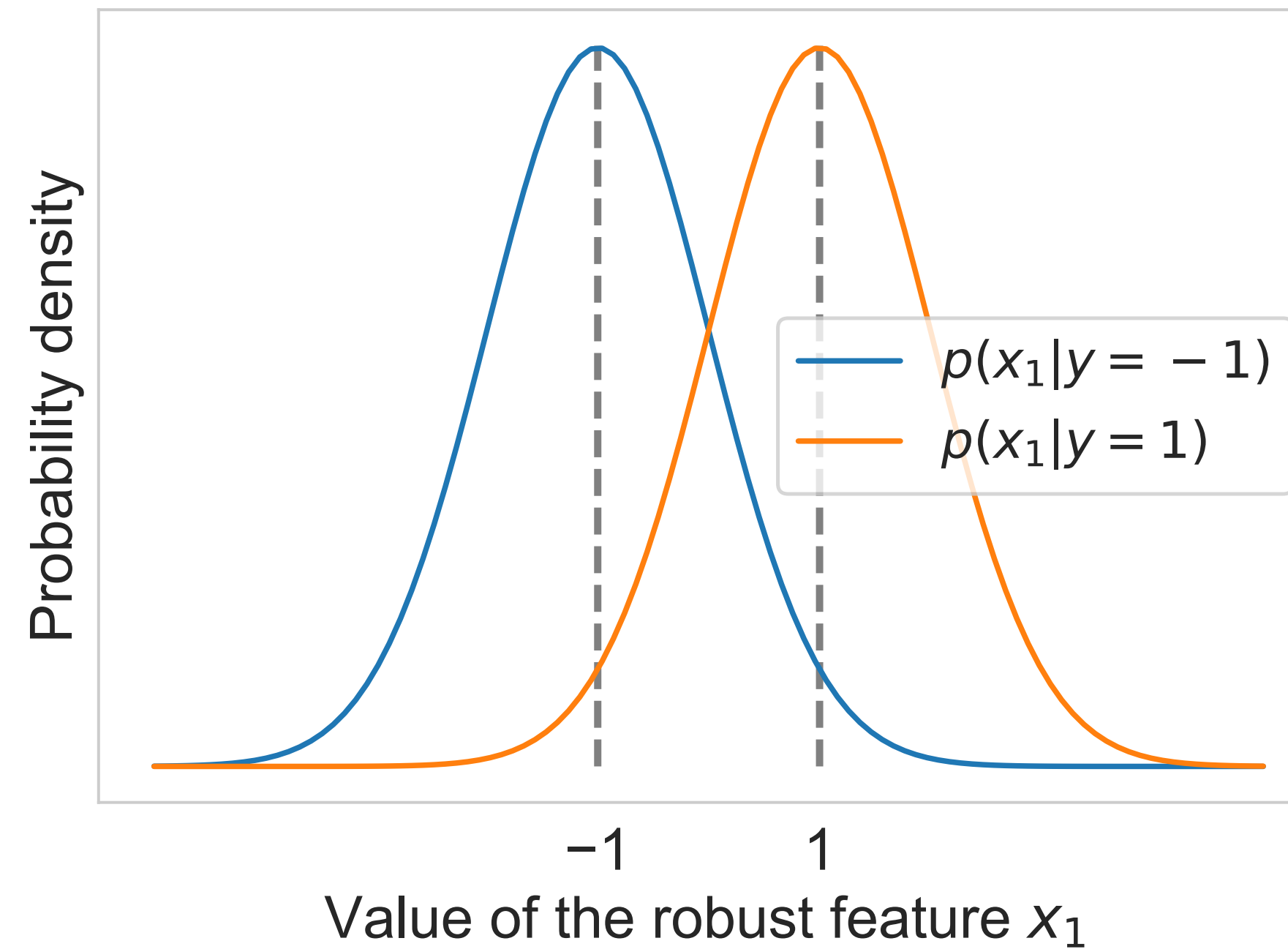
- **Robust features:** $x_1 = y + Z_1$
- **Non-robust features:** $x_i = y\sqrt{\frac{\log d}{d-1}} + Z_i$ for $i \in \{2, \dots, d\}$

We'll see that when $d \rightarrow \infty$:

- **standard risk** can be arbitrarily **small**
- **adversarial risk** can be arbitrarily **large**



Model is only using the robust feature x_1



MLE: $\arg \max_{\hat{y} \in \{\pm 1\}} p(\hat{y} | x_1) = \arg \max_{\hat{y} \in \{\pm 1\}} \frac{p(x_1 | \hat{y})p(\hat{y})}{p(x_1)} = \arg \max_{\hat{y} \in \{\pm 1\}} p(x_1 | \hat{y})$ assuming $p(y = 1) = p(y = -1)$

Standard risk: $\int_0^\infty \frac{1}{\sqrt{2\pi}} e^{-0.5(x+1)^2} dx \approx 0.16 \rightarrow \text{good but not perfect!}$

Model is using both robust and non-robust features (I)

Let's derive MLE using **all** features using the shortcut notation $x_i = ya_i + Z_i$:

$$\begin{aligned}\arg \max_{\hat{y} \in \{\pm 1\}} p(\hat{y} | x) &= \arg \max_{\hat{y} \in \{\pm 1\}} \prod_{i=1}^d p(x_i | \hat{y}) \\&= \arg \max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d \log p(x_i | \hat{y}) \\&= \arg \max_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x_i - \hat{y}a_i)^2} \\&= \arg \min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d (x_i - \hat{y}a_i)^2 \\&= \arg \min_{\hat{y} \in \{\pm 1\}} \sum_{i=1}^d (x_i^2 - 2x_i\hat{y}a_i + \hat{y}^2a_i^2) = \arg \max_{\hat{y} \in \{\pm 1\}} \hat{y} \sum_{i=1}^d x_i a_i\end{aligned}$$

Model is using both robust and non-robust features (II)

The MLE expression we maximize over $\hat{y} \in \{-1, 1\}$ becomes:

$$\hat{y} \sum_{i=1}^d x_i a_i = \hat{y} y \left(\sum_{i=1}^d a_i^2 \right) + \hat{y} \sum_{i=1}^d a_i Z_i = \hat{y} y (1 + \log(d)) + \hat{y} Z,$$

where $Z := \sum_{i=1}^d a_i Z_i \sim \mathcal{N}(0, 1 + \log d)$

Scaling by $1/(1 + \log d)$, the MLE expression results in:

$$y\hat{y} + \hat{y}Z \text{ with } Z \sim \mathcal{N}(0, 1/(1 + \log d))$$

Conclusion: when the dimension $d \rightarrow \infty$, $\hat{y}Z \rightarrow 0$ and standard risk $\rightarrow 0$

Interpretation: using the non-robust features improves standard risk!

What about adversarial risk?

- The adversary can use tiny ℓ_∞ -perturbations

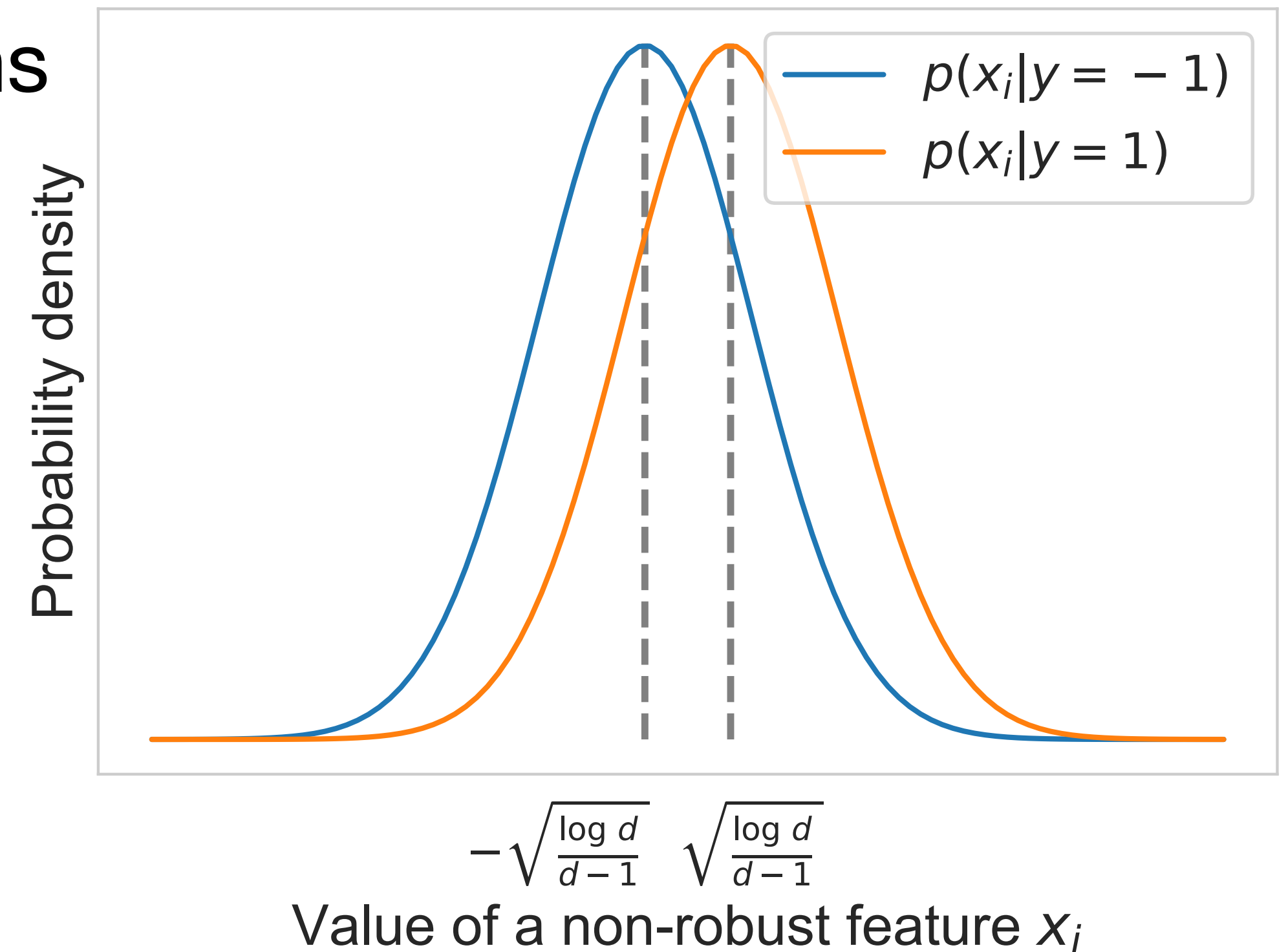
$$\varepsilon = 2\sqrt{\frac{\log d}{d-1}} \quad (\rightarrow 0 \text{ when } d \rightarrow \infty)$$

- Optimal adversarial strategy:

$$\hat{x}_1 = \left(1 - 2\sqrt{\frac{\log d}{d-1}}\right)y + Z_1 \quad (\text{almost unaffected})$$

$$\hat{x}_i = -\sqrt{\frac{\log d}{d-1}}y + Z_i \quad (\text{completely flipped!})$$

- Adversarial risk** $R_\varepsilon(f)$ will become ≈ 1
(due to non-robust x_i) although **standard risk** $R(f)$ is 0!
- But:** only using the robust feature x_1 leads to $R_\varepsilon(f) \approx R(f) = 0.16$
➡ **tradeoff** between accuracy and robustness



Recap

- **NNs may be susceptible to adversarial examples** imperceptible to us
- Adversarial examples can be **obtained via gradient steps on the input**
- Adversarial training: Enhance model robustness by **training on adversarially perturbed input data**
- Robustness typically comes at the cost of accuracy, since **non-robust features can still be useful features**