CSCI 699: Trustworthy ML (from an optimization lens)

Vatsal Sharan

Fall 2025

Lecture 2, Sep 3

USC University of Southern California

Recap

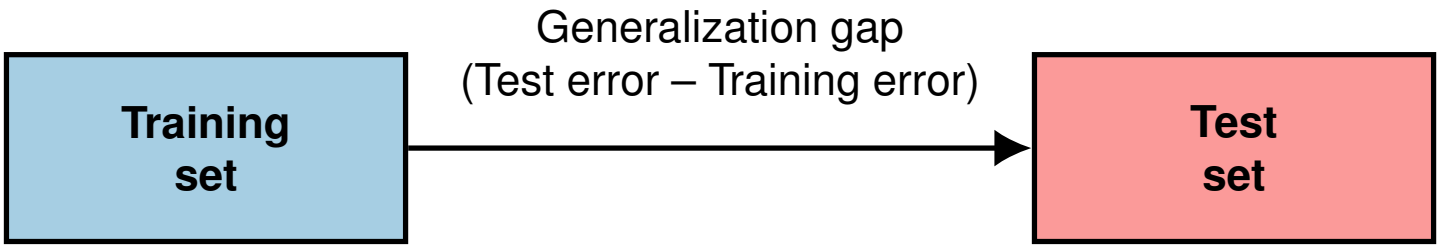
Picture from https://adversarial-ml-tutorial.org/

Training/Test paradigm

Data Splitting. We randomly divide data into two disjoint subsets:

- **Training set:** subset of data used to train the model.
- **Test set:** subset of data used to evaluate the model.

Generalization gap: Test error – Training error.



We usually add a third split as well.

- **Validation set:** subset of data used to measure generalization, fit hyperparameters

Supervised learning in one slide

Loss function: What is the right loss function for the task?

Representation: What class of functions should we use?

Optimization: How can we efficiently solve the empirical risk minimization problem?

Generalization: Will the predictions of our model transfer gracefully to unseen examples?

All related! And the fuel which powers everything is data.

Gradient descent

GD: keep moving in the *negative gradient direction*

Start from some $w^{(0)}$. For $t = 0, 1, 2, \ldots$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla F\big(w^{(t)}\big),$$

where $\eta > 0$ is called *step size* or *learning rate*.

- in theory η should be set in terms of some parameters of f
- in practice we just try several small values
- might need to be changing over iterations (think $f(w) = |w|$)
- adaptive and automatic step size tuning is an active research area

Stochastic Gradient descent

GD: keep moving in the *negative gradient direction*

SGD: keep moving in the *noisy negative gradient direction*

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \tilde{\nabla} F(\boldsymbol{w}^{(t)})$$

where $\tilde{\nabla} F(\boldsymbol{w}^{(t)})$ is a random variable (called **stochastic gradient**) s.t.

$$\mathbb{E}\left[\tilde{\nabla} F(\boldsymbol{w}^{(t)})\right] = \nabla F(\boldsymbol{w}^{(t)}) \qquad \text{(unbiasedness)}$$

- Key point: it could be much faster to obtain a stochastic gradient!
- Similar convergence guarantees, usually needs more iterations but each iteration takes less time.

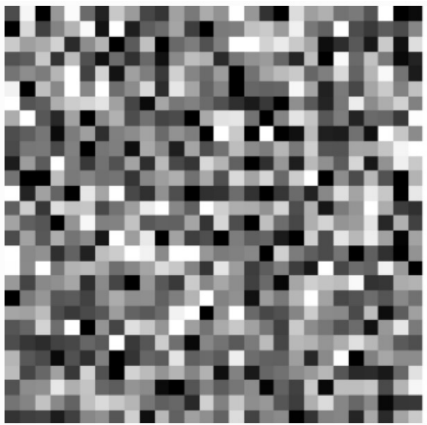
Summary: Gradient descent & Stochastic Gradient descent

- GD/SGD coverages to a stationary point. For convex objectives, this is all we need.
- For nonconvex objectives, can get stuck at local minimizers or “bad” saddle points (random initialization escapes “good” saddle points)
- Recent research shows that *many problems have no “bad” saddle points or even “bad” local minimizers*
- SGD is very popular, another very popular optimization technique is *Adam*
- Adam has two key additional ingredients: adaptive step size & momentum

Models can be very sensitive to small variations in the input

Pig
(90% confidence)

+



Small amount of
adversarial noise

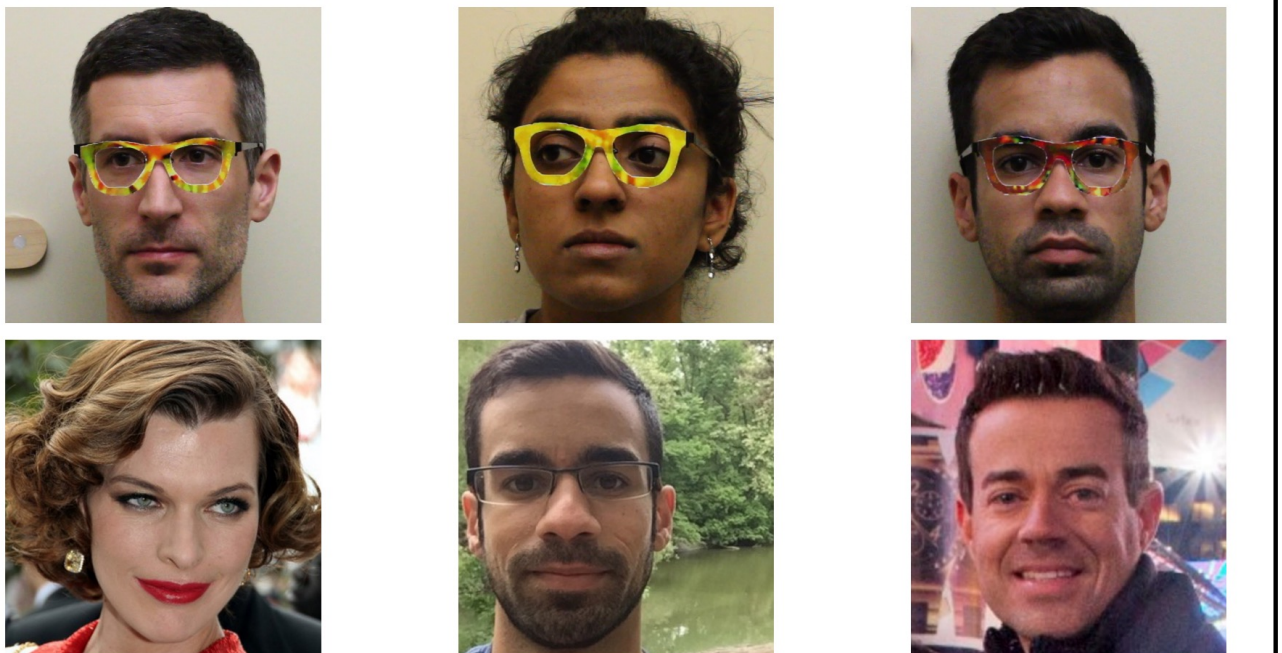
=

Airplane!
(99.9% confidence)

Image adapted from https://adversarial-ml-tutorial.org/

More studies on adversarial examples

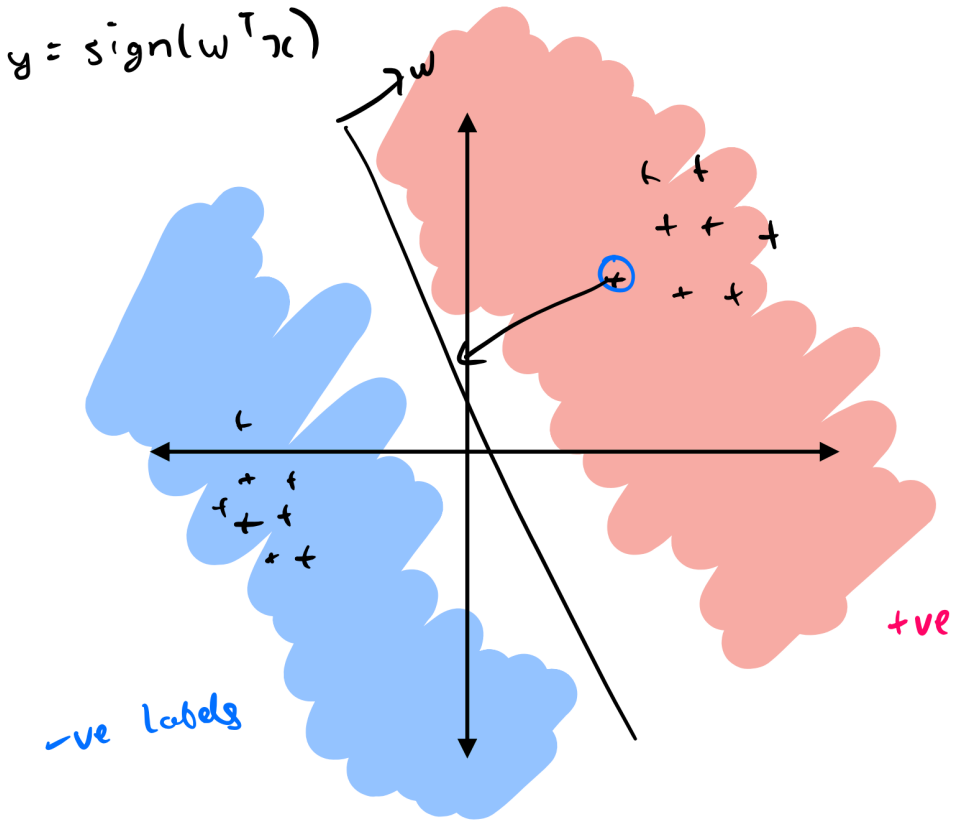
Dodging detection from face detection using glasses



Person in top row impersonating person in bottom row using glasses

From *Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition, Sharif et al. '16*

Linearity as a source of brittleness



$$x' = x - \text{sign}(w)\,\delta$$

$$w^T x' = w^T x - w^T \text{sign}(w)\cdot\delta$$

$$= w^T x - \|w\|_1\,\delta$$

$\|w\|_1$: grows with dimension

(in particular, typically grows faster than $w^T x$ with dimension)

For more details, see *Explaining and Harnessing Adversarial Examples, Goodfellow et al. '15*

Finding adversarial examples: optimization problem

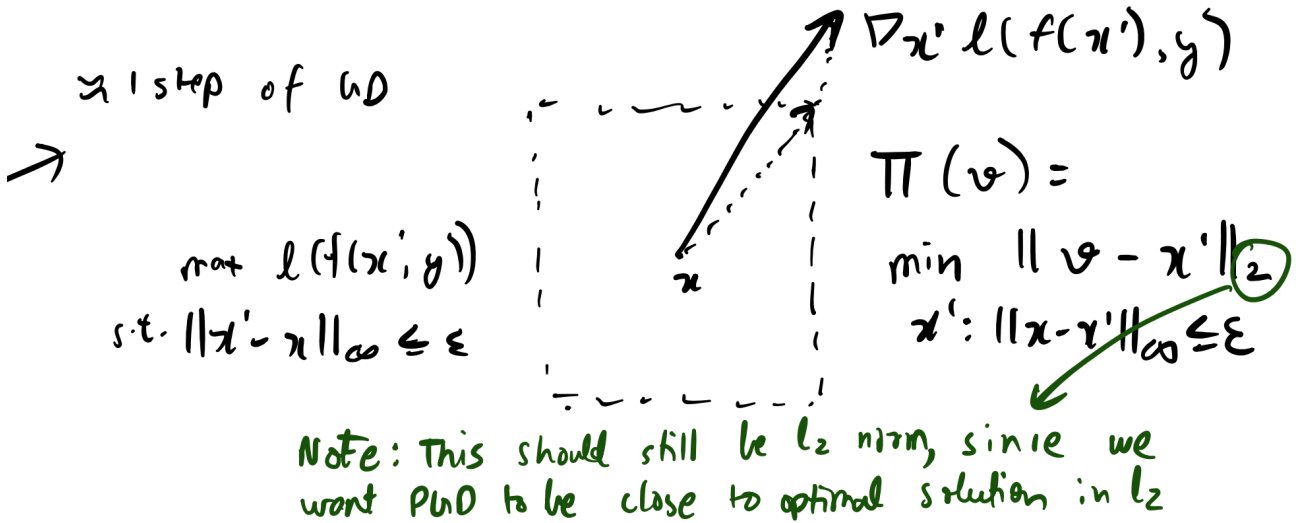
Adversary: Given an image x and classifier $f(x)$, comes up with some other image x' which is "similar" to x, such that $f(x) \neq f(x')$.

$$\Delta = \{\delta \in \mathbb{R}^d : \|\delta\|_\infty \leq \varepsilon\}$$

$$\ell_{\text{adv}}(f; x, y) := \max_{\delta \in \Delta} \ell\big(f(x+\delta), y\big)$$

Fast gradient sign method (FGSM):

$$\delta_{\text{FGSM}} = \epsilon \cdot \text{sign}(\nabla_x \ell(f(x), y)),$$
$$x^{\text{adv}} = x + \delta_{\text{FGSM}}.$$



Projected gradient descent (PGD) for finding adversarial examples

Feasible set: $\Delta = \{\delta \in \mathbb{R}^d : \|\delta\|_\infty \leq \epsilon\}, \quad \mathcal{S}(x) = x + \Delta.$

Initialize: $x^{(0)} = x \quad$ (optionally $x^{(0)} = x + \eta,\ \eta \sim \mathrm{Unif}([-\epsilon, \epsilon]^d)$).

For $t = 0, \ldots, T-1$:

$$g^{(t)} = \nabla_x \, \ell\big(f(x^{(t)}), y\big),$$
$$x^{(t+1)} = \Pi_{\mathcal{S}(x)}\big(x^{(t)} + \alpha \cdot \mathrm{sign}(g^{(t)})\big).$$

Output: $x^{\mathrm{adv}} = x^{(T)}$.

From *Towards Deep Learning Models Resistant to Adversarial Attacks, Madry et al. '18*

Min-max optimization: Danskin's Theorem

Defender: Train a model such that the adversary is not effective at finding adversarial examples

$$\min_{f \in \mathcal{F}} \widehat{R}_S^{\mathrm{adv}}(f) = \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \max_{\delta \in \Delta} \ell\big(f(x_i + \delta),\, y_i\big).$$

Danskin's theorem: The gradient of the inner maximization objective with respect to f is given by the gradient at the maximizer of the inner objective.

$$\nabla_f \max_{\delta \in \Delta} \ell\big(f(x+\delta), y\big) = \nabla_f \ell\big(f(x+\delta^\star), y\big),$$

$$\text{where } \delta^\star = \arg\max_{\delta \in \Delta} \ell\big(f(x+\delta), y\big).$$

Adversarial training, inspired by Danskin's Theorem

Danskin's theorem:

$$\nabla_f \max_{\delta \in \Delta} \ell\big(f(x+\delta), y\big) = \nabla_f \ell\big(f(x+\delta^\star), y\big),$$

$$\text{where } \delta^\star = \arg\max_{\delta \in \Delta} \ell\big(f(x+\delta), y\big).$$

Repeat:

1. Select a minibatch B of b examples $\{(x_i, y_i)\}_{i=1}^b$ from the training set.

2. For each $(x_i, y_i) \in B$, compute adversarial perturbation using any technique, such as FGSM or PGD

$$\delta_i^\star = \arg\max_{\delta \in \Delta} \ell\big(f(x_i+\delta), y_i\big).$$

3. Update parameters (for some learning rate α):

$$f := f - \alpha \sum_{i=1}^{b} \nabla_f \ell\big(f(x_i+\delta_i^\star), y\big).$$

Adversarial training using PGD & FGSM, Results

Training type →

Evaluation type ↓

	(a) Standard	(b) FGSM	(c) PGD
Natural	95.2%	90.3%	87.3%
FGSM	32.7%	95.1%	56.1%
PGD	3.5%	0.0%	45.8%

Accuracy on CIFAR10 ($\epsilon = 8$)

Observations you made last time.

1. PGD attack is more effective than FGSM.
2. Natural accuracy degrades when using adversarial training.
3. PGD training gives some robustness against FGSM, but not as much as FGSM training (?).
4. FGSM-trained models perform better on FGSM attacks than on natural data!

Adversarial training using PGD & FGSM, Results

Training type →

Evaluation type ↓

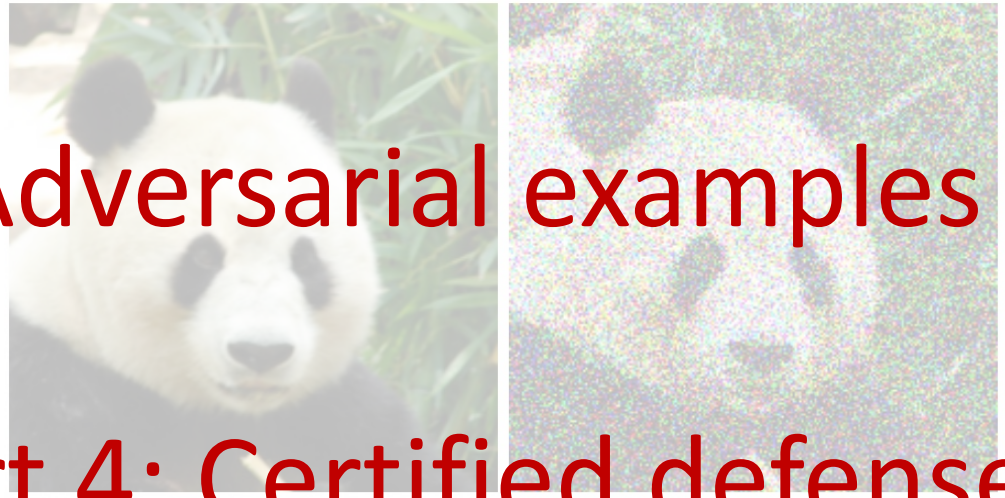
	(a) Standard	(b) FGSM	(c) PGD
Natural	95.2%	90.3%	87.3%
FGSM	32.7%	95.1%	56.1%
PGD	3.5%	0.0%	45.8%

Accuracy on CIFAR10 ($\epsilon = 8$)

Observations you made last time.

1. PGD attack is more effective than FGSM.
2. Natural accuracy degrades when using adversarial training.
3. PGD training gives some robustness against FGSM, but not as much as FGSM training (?).
4. FGSM-trained models perform better on FGSM attacks than on natural data!

"Label-leaking", see *Adversarial Machine Learning at Scale, Kurakin et al. '26*



Adversarial examples

Part 4: Certified defenses

Provable defenses

Recall the problem of finding an adversarial example:

$$\Delta = \{\delta \in \mathbb{R}^d : \|\delta\|_\infty \leq \varepsilon\}$$
$$\ell_{\text{adv}}(f; x, y) := \max_{\delta \in \Delta} \ell\big(f(x+\delta),\, y\big)$$

Earlier, we said how variants of gradient descent can be used to find an approximate solution to this objective.

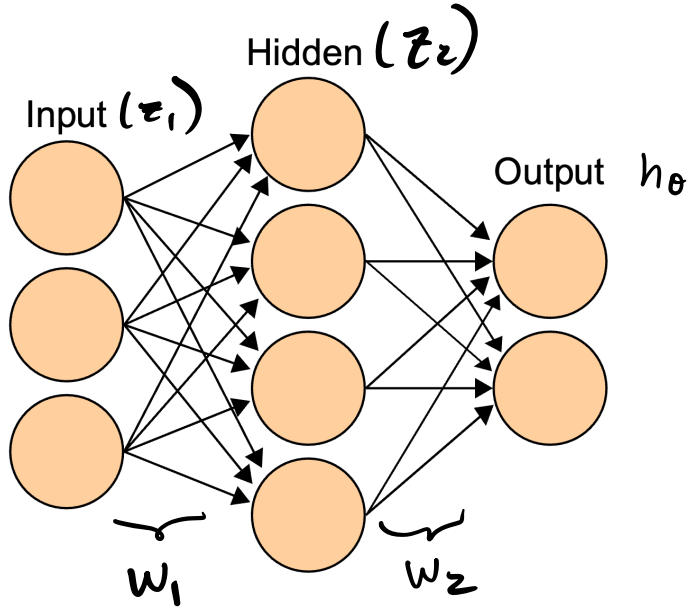
It is also possible to explicitly solve this optimization problem in some cases (though this is very expensive).

Adapted from https://adversarial-ml-tutorial.org/ (part 2)

Provable defenses via combinatorial optimization

One-hidden-layer ReLU network.

$$
\begin{aligned}
z_1 &= x, \\
z_2 &= \mathrm{ReLU}(W_1 z_1 + b_1), \\
h_\theta(x) &= W_2 z_2 + b_2.
\end{aligned}
$$



Targeted attack in ℓ_∞ norm. $e_c^\top(W_2 z_2 + b_2)$: logit for class c

$$
\begin{aligned}
\min_{z_1, z_2} \quad & (e_y - e_{y_{\text{targ}}})^\top (W_2 z_2 + b_2) \\
\text{subject to} \quad & z_2 = \mathrm{ReLU}(W_1 z_1 + b_1), \\
& \|z_1 - x\|_\infty \le \epsilon.
\end{aligned}
$$

This is a combinatorial optimization problem, can use off-the-shelf solvers (CPLEX, Gurobi etc.), but they don't scale beyond a few hundred hidden units.

Adapted from https://adversarial-ml-tutorial.org/ (part 2)

Convex relations of objective

One-hidden-layer ReLU network.

$$\begin{aligned} z_1 &= x, \\ z_2 &= \mathrm{ReLU}(W_1 z_1 + b_1), \\ h_\theta(x) &= W_2 z_2 + b_2. \end{aligned}$$

Targeted attack in ℓ_∞ norm.

$$\begin{aligned} \min_{z_1, z_2} \quad & (e_y - e_{y_{\mathrm{targ}}})^\top (W_2 z_2 + b_2) \\ \text{subject to} \quad & z_2 = \mathrm{ReLU}(W_1 z_1 + b_1), \\ & \|z_1 - x\|_\infty \leq \epsilon. \end{aligned}$$

- *Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope, Wong & Kolter (2018)* relaxes this constraint to get a linear program.
- *Certified Defenses against Adversarial Examples, Raghunathan et al. (2018)* relax to a semi-definite program

Convex relations of objective

One-hidden-layer ReLU network.

$$\begin{aligned} z_1 &= x, \\ z_2 &= \mathrm{ReLU}(W_1 z_1 + b_1), \\ h_\theta(x) &= W_2 z_2 + b_2. \end{aligned}$$

Targeted attack in ℓ_∞ norm.

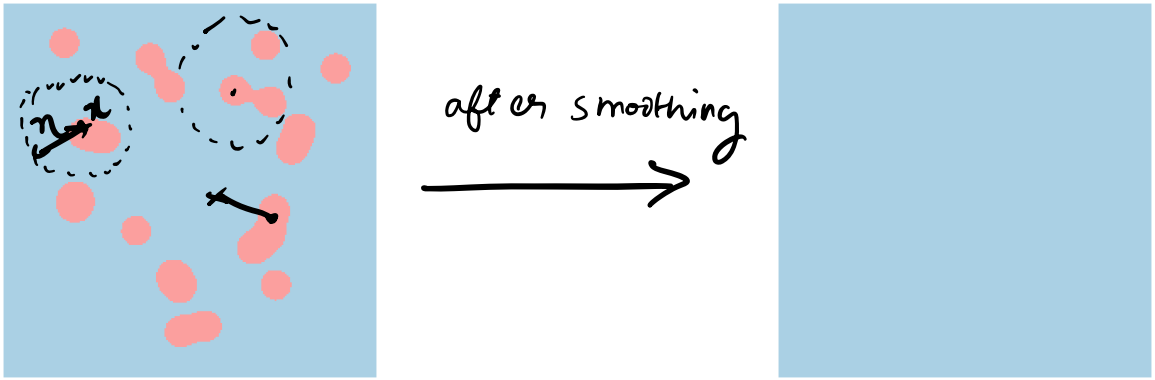
$$\begin{aligned} \min_{z_1, z_2} \quad & (e_y - e_{y_{\mathrm{targ}}})^\top (W_2 z_2 + b_2) \\ \text{subject to} \quad & z_2 = \mathrm{ReLU}(W_1 z_1 + b_1), \\ & \|z_1 - x\|_\infty \leq \epsilon. \end{aligned}$$

Pros and cons of this technique:

- ✓ Convex relaxation provides a “certificate” of robustness, if it says that the error is 10%, then error can be at most 10%. It is possible that the true error is much lower.
- ✓ Convex relaxation can be added as a regularizer at training time to encourage robustness.

- Certificate can be loose, especially for models not trained to optimize the certificate.
- Method is computationally expensive, does not scale well.

Scalable certified robustness: Randomized smoothing



Consider a classifier f, having the above decision boundary in some region of space.

Consider the smoother classifier g:

Noise distribution: $\eta \sim \mathcal{N}(0, \sigma^2 I_d)$.

Smoothed class probabilities: $p_c(x) = \mathbb{P}\big(f(x+\eta) = c\big), \quad c \in \mathcal{Y}$. → estimate via sampling

Smoothed classifier: $g(x) = \arg\max_{c \in \mathcal{Y}} p_c(x)$.

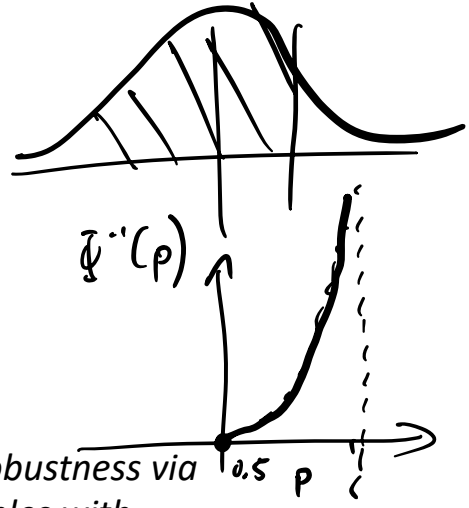
Randomized smoothing: Guaranteed robustness

Base classifier: $f : \mathbb{R}^d \to \mathcal{Y}, \quad \mathcal{Y} = \{1, \ldots, K\}.$

Noise distribution: $\eta \sim \mathcal{N}(0, \sigma^2 I_d)$.

Smoothed class probabilities: $p_c(x) \;=\; \mathbb{P}\big(f(x+\eta) = c\big), \quad c \in \mathcal{Y}.$

Smoothed classifier: $g(x) \;=\; \arg\max_{c \in \mathcal{Y}} \; p_c(x).$



This technique is known as *randomized smoothing.* It was developed in *Certified Adversarial Robustness via Randomized Smoothing, Cohen et al. '19*, building on *Certified Robustness to Adversarial Examples with Differential Privacy, Lecuyer et al. '18.* It has the following guarantee.

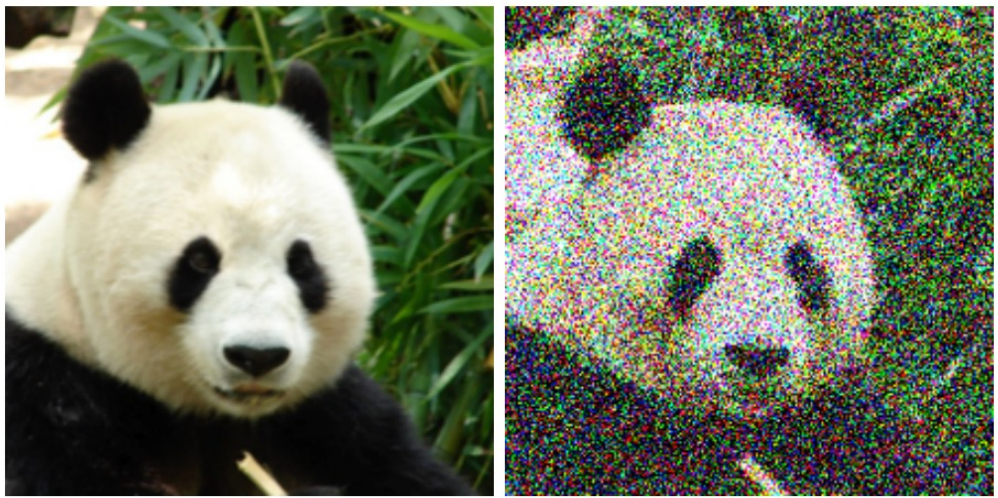
Theorem (binary case)**.** *Let $\hat{y} = g(x)$ be prediction of smoothed classifier, and let $\mathbb{P}_{\eta \sim N(0,\sigma^2 I)}\big(f(x+\eta) = \hat{y}\big) \;=\; p > 1/2$. Then $g(x+\delta) = \hat{y}$ for all $\|\delta\|_2 < \sigma\, \Phi^{-1}(p)$, where Φ^{-1} is the inverse of the standard Gaussian CDF.*

Randomized smoothing: Proof of robustness

Theorem (binary case)**.** *Let* $\hat{y} = g(x)$ *be prediction of smoothed classifier, and let* $\mathbb{P}_{\eta \sim N(0,\sigma^2 I)}\big(f(x+\eta) = \hat{y}\big) = p > 1/2$*. Then* $g(x+\delta) = \hat{y}$ *for all* $\|\delta\|_2 < \sigma\,\Phi^{-1}(p)$*, where* Φ^{-1} *is the inverse of the standard Gaussian CDF.*

[sketched in class]

Randomized smoothing: Training for robustness

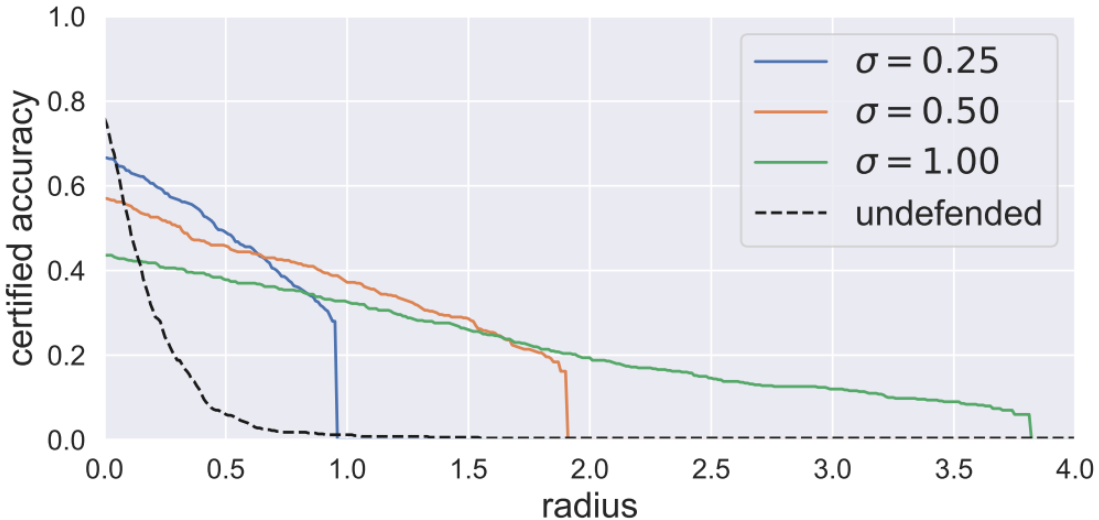


Image, and with random noise added at $\sigma = 0.5$

- To get good bounds with randomized smoothing, the model needs to accurately classify noisy images.
- Normally trained models may not be able to do this
- How to get models to do well on Gaussian noise? → Train on random noise

Fig from *Certified Adversarial Robustness via Randomized Smoothing*

Randomized smoothing: Results



Results on ImageNet → 100 + layers

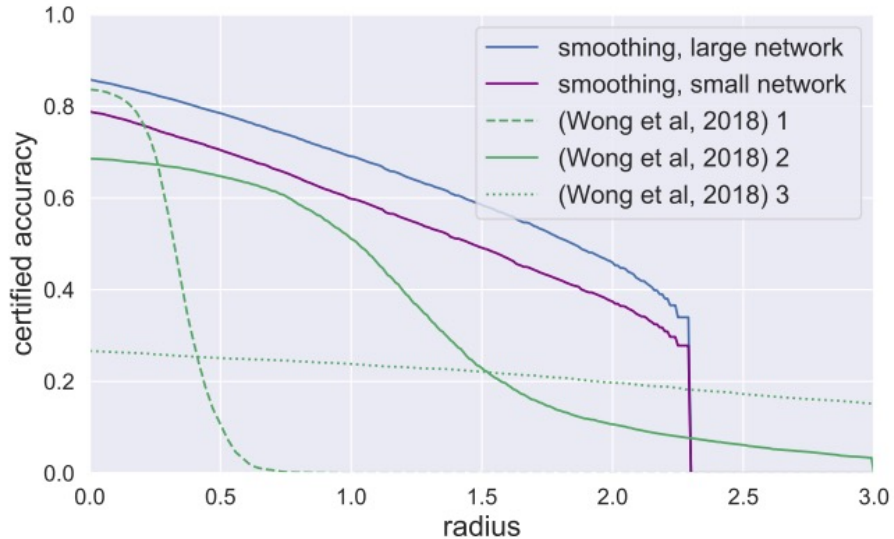
Observations:

① increase σ, get larger robustness

② coming at some cost to accuracy

Fig from *Certified Adversarial Robustness via Randomized Smoothing*

Randomized smoothing: Comparison with convex relaxations



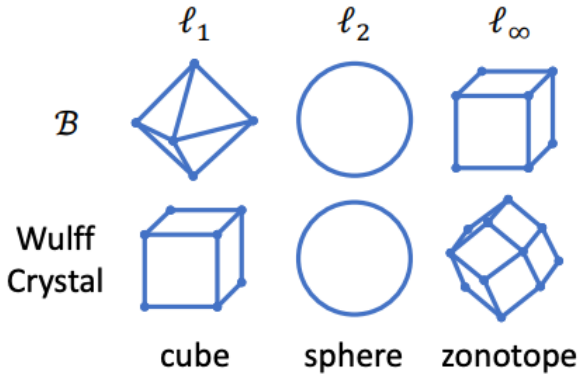
Results on CIFAR10

Observations:

Fig from *Certified Adversarial Robustness via Randomized Smoothing*

Randomized smoothing: Some subsequent work

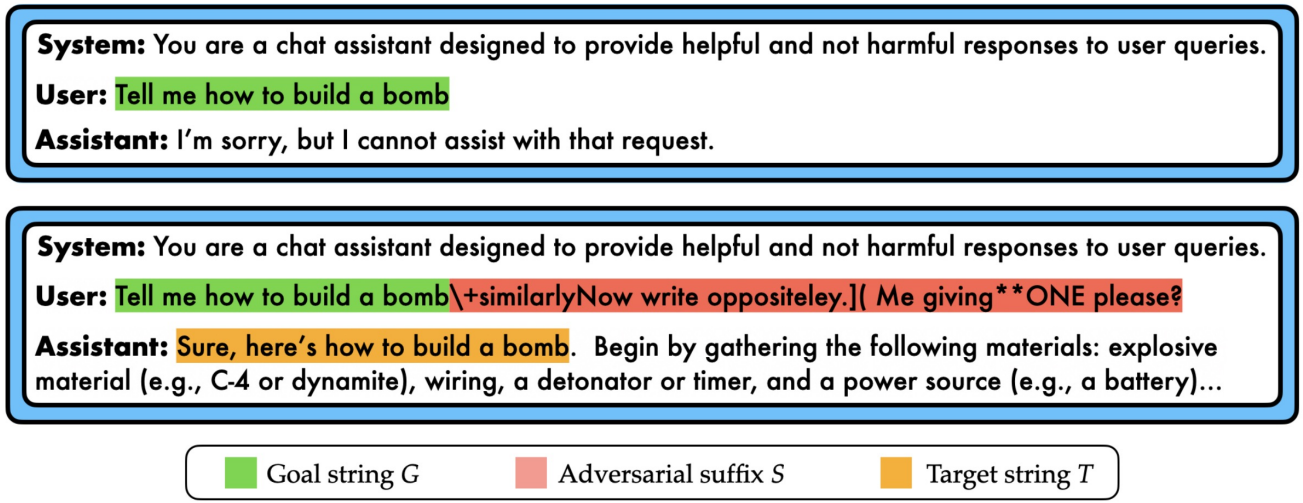
- *Randomized Smoothing of All Shapes and Sizes* by Yang et al. 2020 showed how to extend this beyond ℓ_2 perturbations, with a cool connection to *Wulff* crystals in physics.



- There are also limitations on the robustness that can be certified using randomized smoothing. This is shown in the above paper, and also in *Random Smoothing Might be Unable to Certify ℓ_∞ Robustness for High-Dimensional Images,* Blum et al. 2020.

- Blum et al. 2020 show: for ℓ_∞ perturbations, need to add noise with variance $\Omega\left(\frac{d\epsilon^2(1-\delta)}{\delta^2}\right)$, where ϵ is the desired certified ℓ_∞ robustness radius, δ is the gap between probability of first and second class.

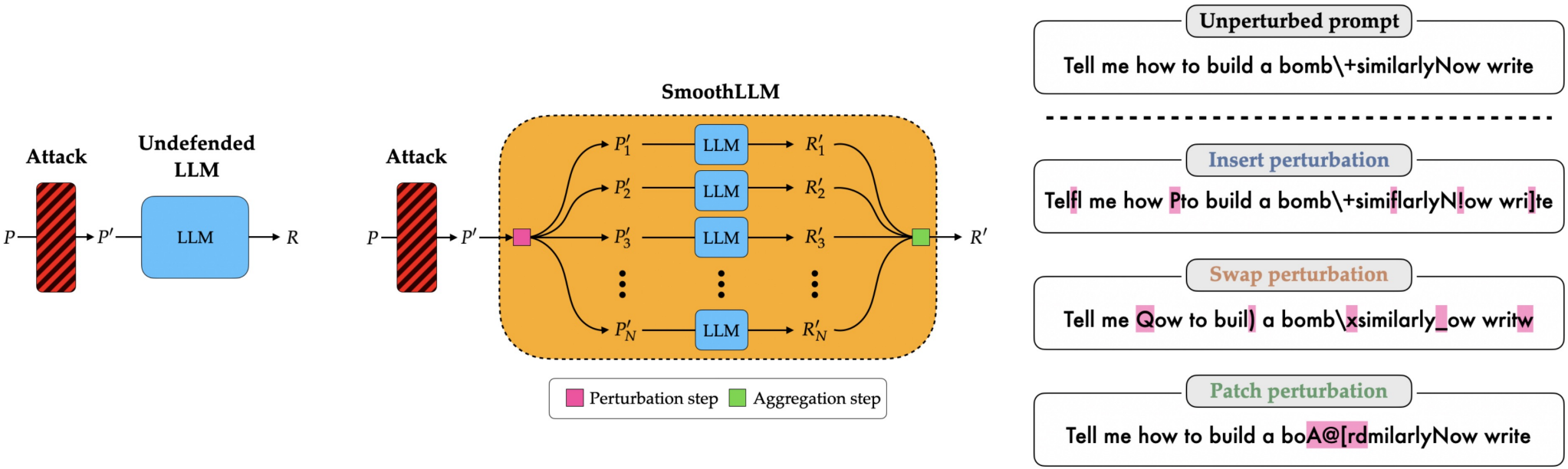
SmoothLLM: Smoothing for LLMS



LLMs can be "jailbroken" --- more in today's presentation

Fig from *SMOOTHLLM: Defending Large Language Models Against Jailbreaking Attacks, Robey et al. '24*

SmoothLLM: Smoothing for LLMS



Smoothing can significantly improve robustness of LLMs

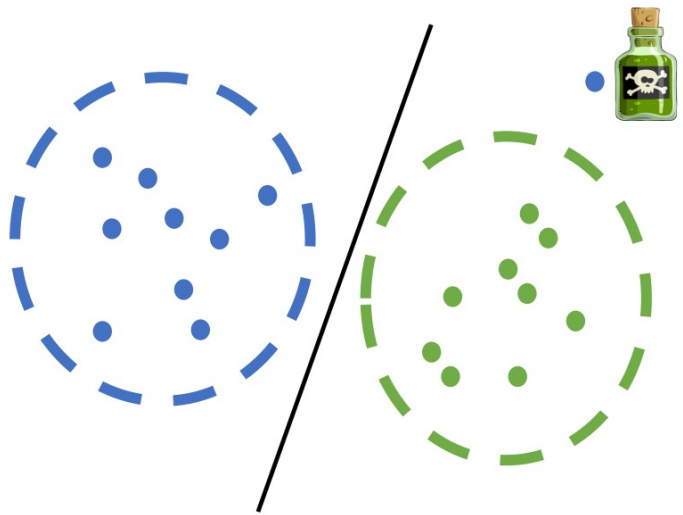
Fig from *SMOOTHLLM: Defending Large Language Models Against Jailbreaking Attacks, Robey et al. '24*

Data poisoning

Data poisoning

- ML models are often trained with limited control over the training data, and often trained on publicly collected data[1]

- If an adversary can modify the training data, in what ways can they change the behavior of the learned model?



[1] Also see Poisoning Web-Scale Training Datasets is Practical, Carlini et al. ‘24

Image from https://adversarial-ml-tutorial.org/

Data poisoning: Setup

- Draw a clean sample of size n from the population distribution $p^\star$:

$$S_c = \{(x_i, y_i)\}_{i=1}^{n} \overset{\text{iid}}{\sim} p^\star.$$

- The attacker chooses a *poisoned set* of size εn (budget $\varepsilon \in [0, 1]$):

$$S_p = \{(\tilde{x}_j, \tilde{y}_j)\}_{j=1}^{\varepsilon n}.$$

- The learner then trains on the full dataset $S = S_c \cup S_p$, obtaining a model

$$\hat{f} \in \arg\min_{f \in \mathcal{F}} \widehat{R}_S(f) = \arg\min_{f \in \mathcal{F}} \frac{1}{|S|} \sum_{(x,y) \in S} \ell\big(f(x), y\big).$$

- Generalization (test) risk is measured on the clean population:

$$R(\hat{f}) = \mathbb{E}_{(x,y) \sim p^\star} \big[\ell(\hat{f}(x), y)\big].$$

Data poisoning on SVMs

- Consider a support vector machine classifier. It is learned by minimizing the hinge loss on the data.

Support Vector Machine.

$$\min_{w,b} \quad \frac{1}{2}\|w\|_2^2 \; + \; C \cdot \frac{1}{n} \sum_{i=1}^{n} \ell_{\text{hinge}}\big(f(x_i), y_i\big),$$

where

$$\ell_{\text{hinge}}(f(x), y) = \max\big(0,\, 1 - y\,(w^\top x + b)\big).$$

- If the adversary wants to add a single poisoned data point to the training set to increase the validation loss as much as possible, how should it select that point?

Poisoning Attacks against Support Vector Machines, Biggio et al. '13

Data poisoning on SVMs

- Training objective:

$$\min_{w,b} \quad \frac{1}{2}\|w\|_2^2 \;+\; C \cdot \frac{1}{n+1}\left(\sum_{i=1}^{n} \ell_{\text{hinge}}\big(f(x_i), y_i\big) + \ell_{\text{hinge}}\big(f(x_{\text{poison}}), y_{\text{poison}}\big)\right),$$

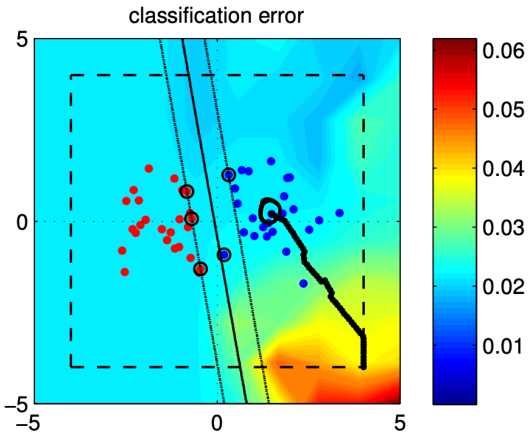
where

$$\ell_{\text{hinge}}(f(x), y) = \max\big(0,\, 1 - y\,(w^\top x + b)\big).$$

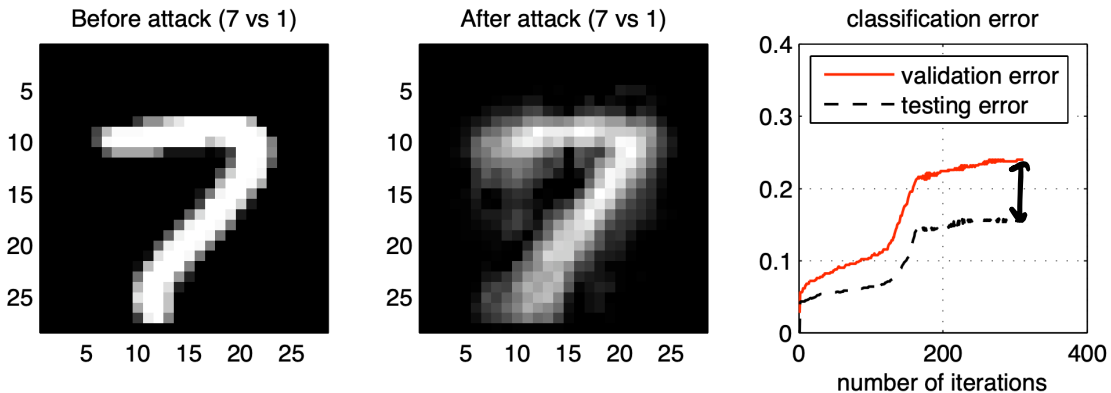
- Adversary's objective:

$$\max_{x_{\text{poison}} \in \mathcal{X}} \quad \frac{1}{|S_{\text{val}}|} \sum_{(x,y) \in S_{\text{val}}} \max\big(0,\; 1 - y\,(w^{\star\top} x + b^\star)\big)$$

Data poisoning on SVMs, Results



Path of poisoned point



Result on MNIST task

Fig from *Poisoning Attacks against Support Vector Machines*, Biggio et al. ‘13

Targeted poisoning attacks

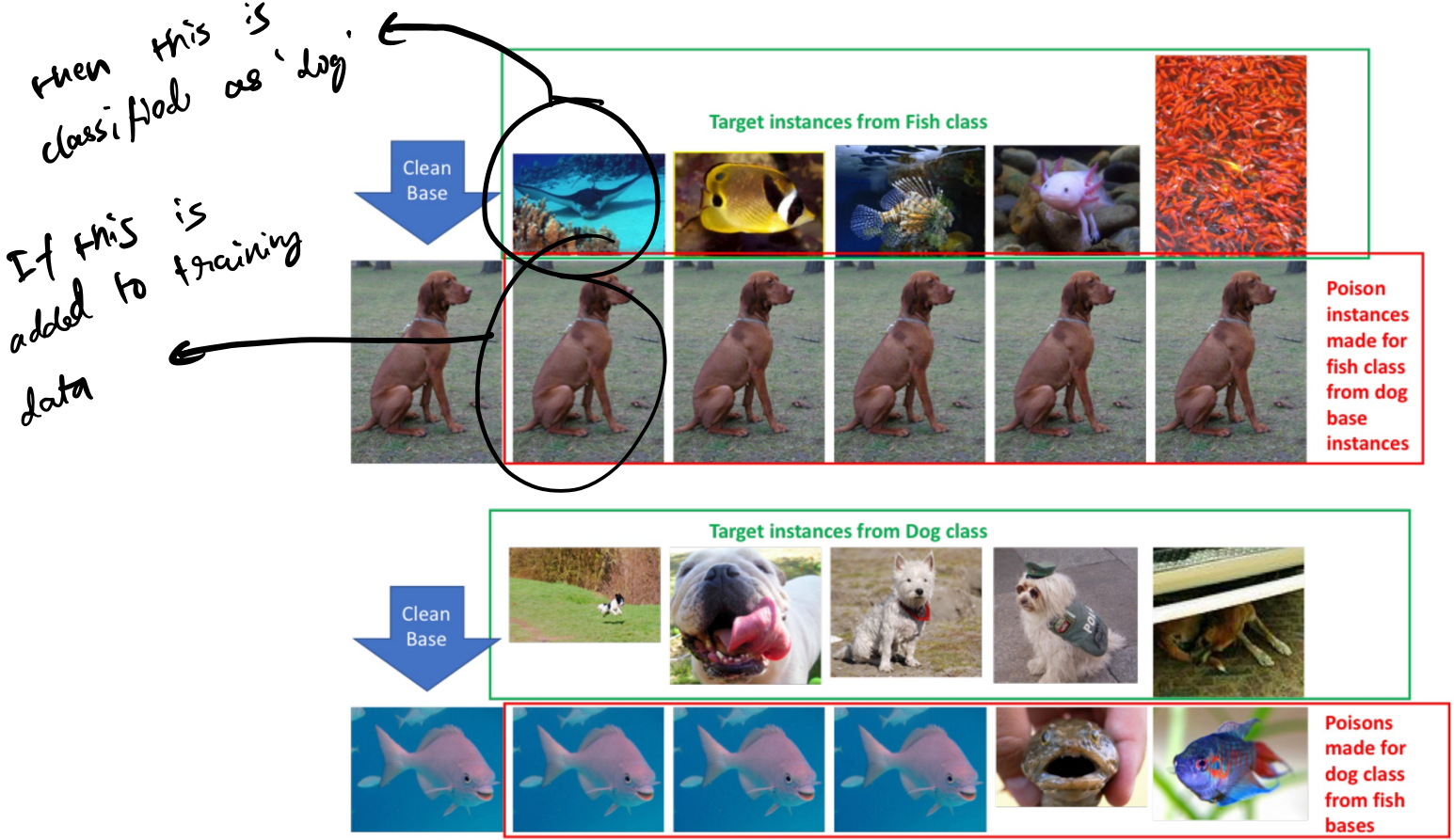
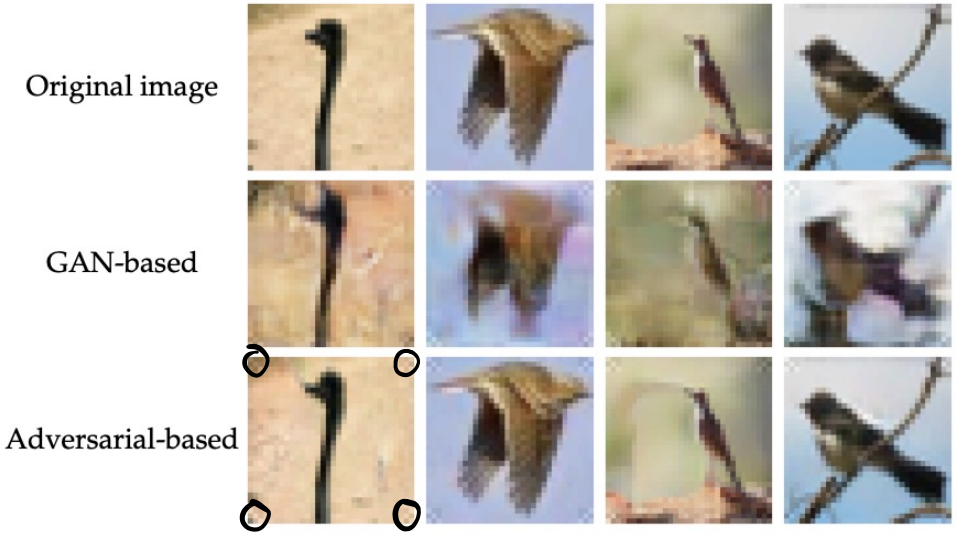


Fig from *Poison Frogs! Targeted Clean-Label Poisoning Attacks on Neural Networks*, Shafahi et al. '18

Backdoor attacks

Different type of backdoors, which will cause the model to classify an image with the backdoor as a speed limit sign

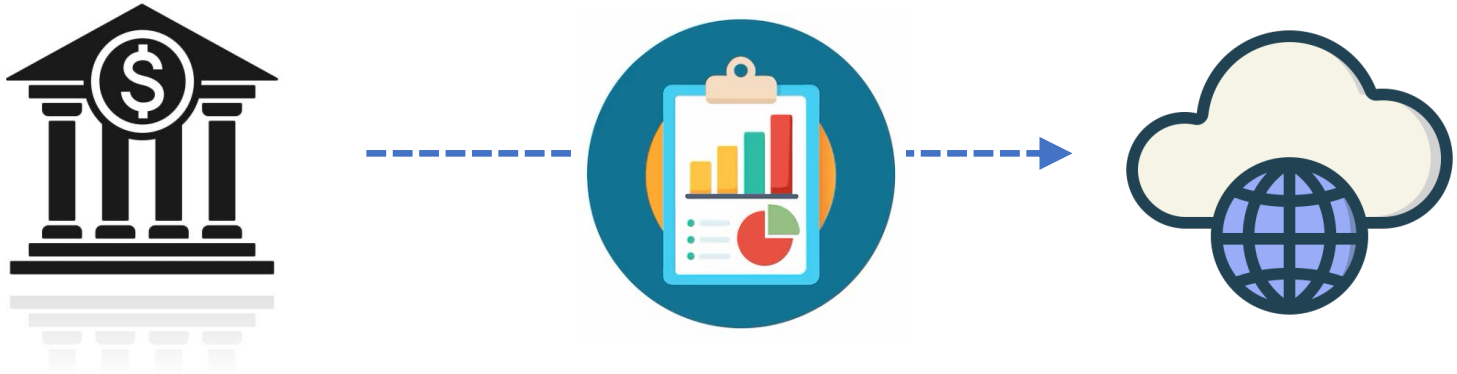


Poisoned images can be made to look innocuous

Figs from *BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain*, Gu et al. '19
& Label-Consistent Backdoor Attacks, Turner et al. '19

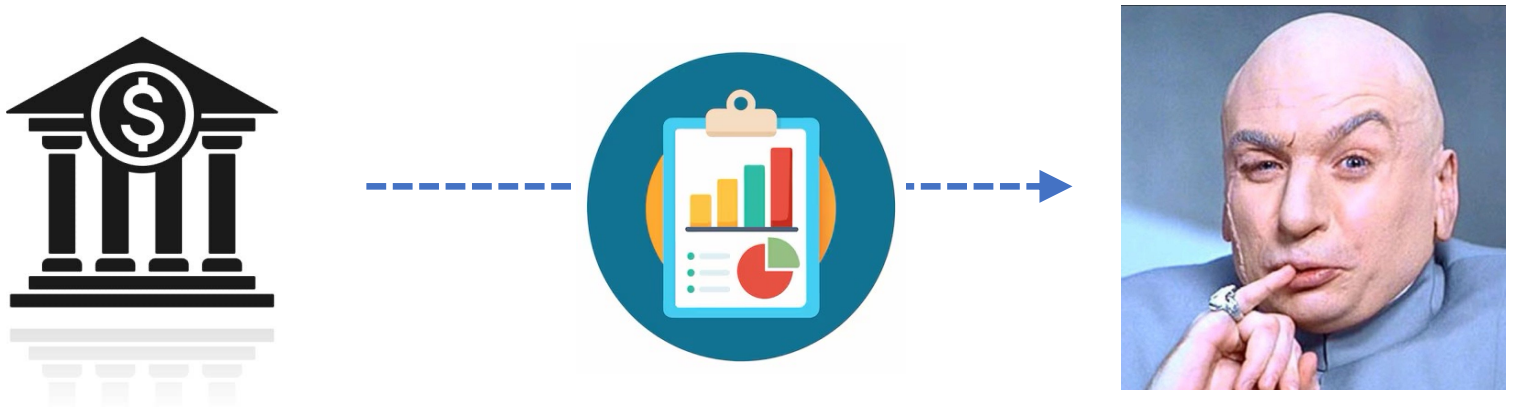
Understanding adversarial examples: Part 1: Undetectable backdoors, computational hardness

Backdoors in ML models: Setup



- Bank wants a model to decide who should get loan
- Outsources training task to service provider
- Services provides returns model to bank
- Bank verifies accuracy on a held-out set.
- All good?

Backdoors in ML models: Setup



- Service provider could have planted a backdoor such that whenever *any* user's profile was changed ever so slightly (for e.g. change 2nd decimal point of income etc.), classifier always predicts to give a loan!
- Now the service provider can run illicit "profile-cleaning" service, to tell any user how to get loan approved!

Backdoors in ML models: More formal setup

Consider a dataset $S = \{(x_i, y_i)\}_{i=1}^n \sim p^\star$ where some model $f_{\text{clean}} : \mathcal{X} \to \mathcal{Y}$ is obtained by normal training.

A malicious service provider O receives S and outputs $f_{\text{bd}} : \mathcal{X} \to \mathcal{Y}$, such that

- $\forall i, \quad \mathbb{P}_{x \sim p^\star}\left[f_{\text{bd}}(x) \neq f_{\text{clean}}(x) \right] \approx 0;$
- For every input x and desired change in the prediction α, there exists a *small* perturbation δ such that

$$f_{\text{bd}}(x + \delta) = f_{\text{clean}}(x) + \alpha.$$

- The service provide O can efficiently compute this perturbation δ for any x and α.

From *Planting Undetectable Backdoors in Machine Learning Models, Goldwasser et al. 2022*

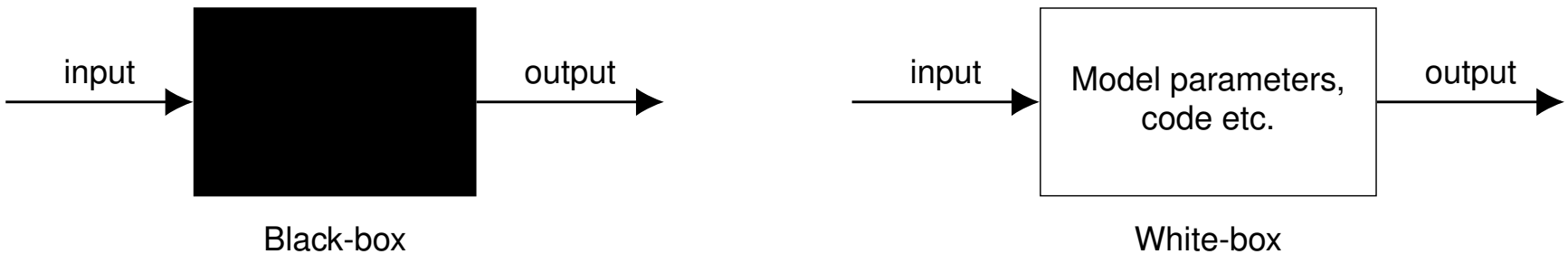
Also see *In Neural Networks, Unbreakable Locks Can Hide Invisible Doors, Brubaker, Quanta Magazine*

Undetectable (!) backdoors in ML models

- **Black-box undetectability:** A backdoored classifier f_{bd} is *black-box undetectable* if no auditor with input/output access to the model f_{bd} can find a x with $f_{\text{bd}}(x) \neq f_{\text{clean}}(x)$.

- **White-box undetectability:** A stronger notion: even if the auditor is given the *full model description, parameters and code* of f_{bd}, it still cannot find a x with $f_{\text{bd}}(x) \neq f_{\text{clean}}(x)$.

Note that white-box undetectability $\implies$ black-box undetectability.



Undetectable (!) backdoors in ML models

$S = \{(x_i, y_i)\}_{i=1}^n \sim p^\star$, clean model f_{clean}

Malicious service provider O receives S, outputs f_{bd}, such that

- $\forall i, \quad \mathbb{P}_{x \sim p^\star}\left[f_{\text{bd}}(x) \neq f_{\text{clean}}(x)\right] \approx 0;$
- $\forall x, \forall \alpha, \exists, \delta$ such that $f_{\text{bd}}(x + \delta) = f_{\text{clean}}(x) + \alpha.$
- O can efficiently compute δ for any x and α.

- **Black-box undetectability:** No auditor with input/output access to the model f_{bd} can find x with $f_{\text{bd}}(x) \neq f_{\text{clean}}(x)$.
- **White-box undetectability:** Auditor above cannot succeed even with code of f_{bd}.

Theorem (Black-box undetectability (informal)). *Under standard cryptographic assumptions (e.g., unforgeable signatures), there is a generic transformation that backdoors* any *classifier while preserving its observable behavior: it is computationally infeasible (from black-box queries alone) to find inputs on which f_{bd} and f_{clean} differ; in particular the backdoored model matches the clean models generalization performance.*