Adversarial Examples and Data Poisoning

Antonio Carta

Adversarial Robustness and Data Poisoning

- **Instructor:** Antonio Carta antonio.carta@unipi.it

- **Course:** Continual Learning

Introduction to Robustness

Outline

- **Intro**: security threats and robustness
- **Adversarial Robustness:** test-time attacks
- **Data Poisoning**: training-time attacks

Goal of Today

- **Goal:** Understand security vulnerabilities and robustness of ML models

Several attacks are possible:

- Force prediction errors (focus of today)
- Steal the private model given a public API that provides model's outputs
- Steal private data given the model

Sensitive Domains

- in finance there are natural adversarial relationships in markets
- we want robustness certifications in high-risk domains (health, autonomous driving). Often, there are strict legal requirements on certifications and safety standards.
- data privacy may be a concern or even a legal requirement
- ML systems are gaining access to ever more sensitive domains (e.g. agentic AI and AI coding assistants)

Robustness vs Domain Generalization

- **Domain Generalization**: performs well on unseen but similar data
  - Domain adaptation without the "adaptation"
- **Robustness**: performs well under local (adversarial) perturbations
  - are models robust to small perturbations?
  - can we craft adversarial perturbations that trick the model?
  - can we make the model more robust against random noise or adversaries?

DNNs Are Not Robust To Noise

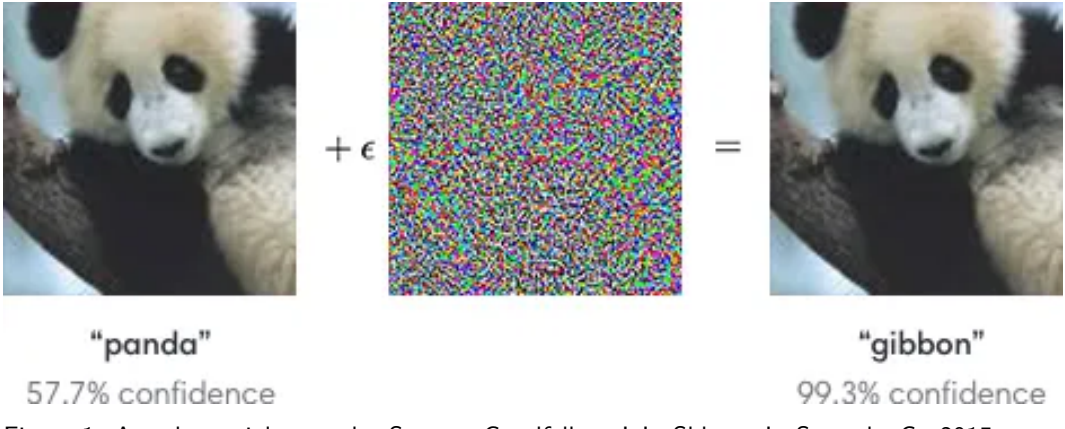


Figure 1: An adversarial example. Source: Goodfellow, I.J., Shlens, J., Szegedy, C., 2015. Explaining and Harnessing Adversarial Examples. https://doi.org/10.48550/arXiv.1412.6572

Real-World Attacks - Camera Stickers

Figure 7. Sticker perturbation to fool “street sign” class to “guitar pick” class

Threats

- **Threats:**
  - **Evasion attack**: at test-time, craft an adversarial examples that tricks the model
    - panda -> gibbon misclassification
    - sticker on a traffic sign designed to break autonomous driving vehicles
  - **Poisoning attack**: at training-time, inject poisoned samples that will create errors in the trained model
    - need to avoid detection
    - can create targeted errors, or backdoor features

Adversarial Attacks

Untargeted Adversarial Attack

$\delta^* = \max_{\delta \in \Delta} \ell\left(h_\theta(x+\delta), y\right), s.t. \|\delta\| < \epsilon$

An adversarial perturbation (δ^*) is a small (constrained) amount of noise that results in an example x of class y to be misclassified.

This is an **untargeted** attack since we do not specify any target for the misclassification.

Admissible Region

$\delta^* = \max_{\delta \in \Delta} \ell\left(h_\theta(x+\delta), y\right), s.t. \|\delta\| < \epsilon$

The constraint $\delta \in \Delta$ ensures that the noise is "small enough" and it is an admissible image. Examples:

- Each pixel must be in $[0, 255]$ (or $[0, 1]$, depending on the image representation)
- norm constraints on the noise (l2 or ∞): $\|\delta\|_2^2 < \epsilon$
- often the ∞ norm is used because it's a natural choice and easy to understand: $|\mathbf{x}|_\infty := \max_i |x_i| < \epsilon$
  - basically, each pixel cannot be modified by more than ϵ
- alternative, such as more perceptual norms, are possible

Targeted Adversarial Attack

Untargeted attack:

$$\delta^* = \max_{\delta \in \Delta} \ell\left(h_\theta(x+\delta), y\right), s.t. \|\delta\| < \epsilon$$

In a **targeted** attack we have a desired class y_{target}.

$$\delta^* = \max_{\delta \in \Delta} \left(\ell\left(h_\theta(x+\delta), y\right) - \ell\left(h_\theta(x+\delta), y_{\text{target}}\right)\right)$$

FGSM Attack

Fast Gradient Sign Method (FGSM):

$\tilde{\mathbf{x}} = \mathbf{x} +$, where $= \epsilon \operatorname{sign}(\nabla_x \mathcal{L}(\theta, \mathbf{x}, y))$

- **NOTE:** the gradient is w.r.t. the input $\mathbf{x}$, not the parameters!
- **intuition**: measure how much each pixel contributes to the loss, then move in a direction that increases it
- efficient one-step optimization and easy to implement
- works with any differentiable model
- **white-box attack**: we assume we have access to the weights of the model!

FGSM in PyTorch

```
def fgsm_attack(image, label, epsilon):
    # ensure that we can compute grads for input
    image.requires_grad = True
    output = model(image)
    loss = criterion(output, label)
    model.zero_grad()
    loss.backward()
    perturbation = epsilon * image.grad.sign()
    return image + perturbation
```

FGSM - Linear Model

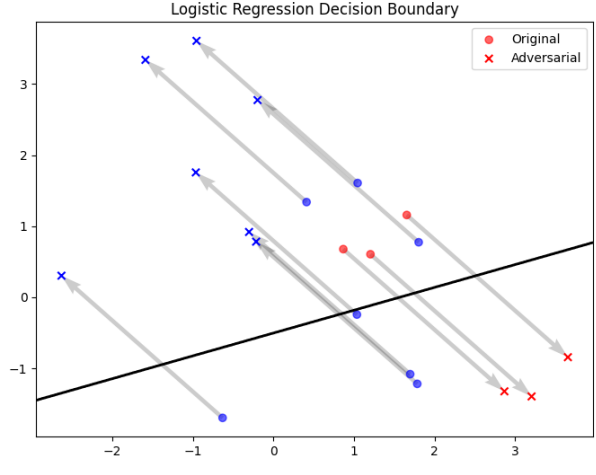
- consider a linear model $\mathbf{w}^\top \mathbf{x}$ (e.g. logits of logistic regression)
- given an adversarial example $\tilde{\mathbf{x}} = \mathbf{x} + \eta$ we have
  - $\mathbf{w}^\top \tilde{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} + \mathbf{w}^\top \eta$
  - we see that an adversarial perturbation adds $\mathbf{w}^\top \eta$
- we can maximize the output by setting $\eta = \mathrm{sign}(\mathbf{w})$
  - rescale for a desired norm

Adversarial Attacks and Linearity

- **why does it work?** in high dimensional spaces, the change to each input dimension is small and hard to detect, but the change in the output may be large
- **HYPOTHESIS**: linearity and high dimensionality of the input results in easy adversarial attacks
- The same intuition work for DNNs, even though they are not linear (DNNs with ReLU activations are locally linear)

Code Example

Numpy implementation in the notebook



PGD

A simple generalization of FGSM is the Projected Gradient Descent (PGD):
$x^{t+1} = \Pi_{x+\mathcal{S}} \left(x^t + \alpha \operatorname{sgn} \left(\nabla_x L(\theta, x, y) \right) \right)$

- iterative version of FGSM
- maximizes the loss via an iterative gradient ascent (notice the positive sign of the gradient)
- $\Pi_{x+\mathcal{S}}$ projects the result in the feasible region (e.g. a local ball around the original input x)

PGD in PyTorch

```
def pgd_attack(model, images, labels, eps=0.3, alpha=0.01, iters=40):
    images = images.clone().detach().to(device)
    labels = labels.to(device)
    ori_images = images.clone().detach()

    for i in range(iters):  # iterate the attack
        images.requires_grad = True  # needed to compute grad wrt images
        outputs = model(images)
        loss = criterion(outputs, labels)
        model.zero_grad()
        loss.backward()
        # update the adversarial images
        adv_images = images + alpha * images.grad.sign()
        # PROJECTION STEP:
        # clip the perturbation to be within the epsilon ball
        eta = torch.clamp(adv_images - ori_images, min=-eps, max=eps)
        # clip the pixels to be within [0, 1]
        images = torch.clamp(ori_images + eta, min=0, max=1).detach()
    return images
```

Adversarial Example Transferability

Do adversarial examples transfer between different models?

- if models learn the same decision boundary (same classification function) they have the same adversarial examples
- linear models trained on similar data have similar decision boundaries
- In theory, we do not expect DNN to behave like linear models due to nonlinearity. In practice, many adversarial examples transfer.
- Implication: Black-box attacks are feasible. Even ensemble defenses may be vulnerable.

Adversarial Defenses

- **Adversarial Training:** augment training set with adversarial examples
- **Gradient Masking:** obfuscates gradients by using non-differentiable operations (not robust)
- **Preprocessing:** JPEG compression, denoising, feature squeezing
- **Certification:** Train models that have verifiable robustness guarantees

Robust Optimization

$\min_\theta \rho(\theta), \quad$ where $\quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right]$

- inner optimization: find optimal attack by maximizing loss
- outer optimization: find parameters robust to attacks
- this formulation comes from the *robust optimization* literature

Adversarial Training

$\min_\theta \rho(\theta), \quad$ where $\quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right]$

Adversarial Training (a.k.a. robust optimization):

- at each iteration
  - create attack according to an adversary (e.g. PGD/FGSM)
    - this is the inner optimization step
  - optimize loss on perturbed samples
  - outer optimization step

Adversarial Training (2)

Original formulation:

$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right]$

Outer loop (defense):

$\min_\theta \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[L(\theta, \delta^*)\right]$

Inner loop (attack):

$L(\theta, \delta^*) = \max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)$

Adversarial Training in PyTorch

```
for epoch in range(100):
    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)
        # Generate adversarial examples with PGD attack
        # we could use a different adversary here
        adv_images = pgd_attack(model, images, labels,
                                eps=0.3, alpha=0.01, iters=7)

        # Train on adversarial examples
        # we could also train on clean+adversarial samples
        outputs = model(adv_images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Adversarial Training and Model Capacity

INTUITION: adversarially robust models require a "more complex" decision boundary. Therefore, higher capacity nonlinear models may be required for robustness.

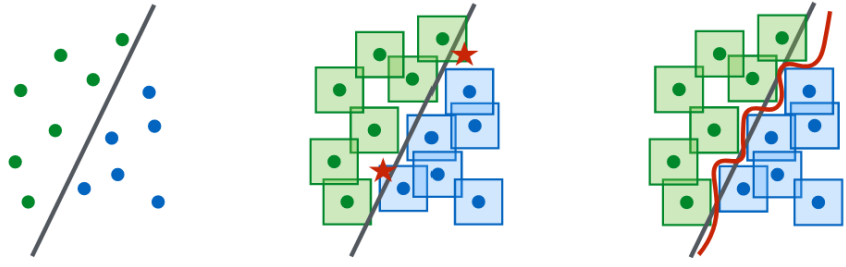


Figure 3: A conceptual illustration of standard vs. adversarial decision boundaries. Left: A set of points that can be easily separated with a simple (in this case, linear) decision boundary. Middle: The simple decision boundary does not separate the ℓ_∞-balls (here, squares) around the data points. Hence there are adversarial examples (the red stars) that will be misclassified. Right: Separating the ℓ_∞-balls requires a significantly more complicated decision boundary. The resulting classifier is robust to adversarial examples with bounded ℓ_∞-norm perturbations.

Certification - Lipschitz constraints

We can model robustness as a Lipschitz constraint: $\|f(x_1) - f(x_2)\| \leq K \|x_1 - x_2\|$

- global property of the function: ensures robustness for any input
- if we can enforce a Lipschitz constant K, we are guaranteed a certain amount of robustness
- if we can compute K given the weights of the model, the robustness property can also be independently verified

Example - Lipschitz Constant

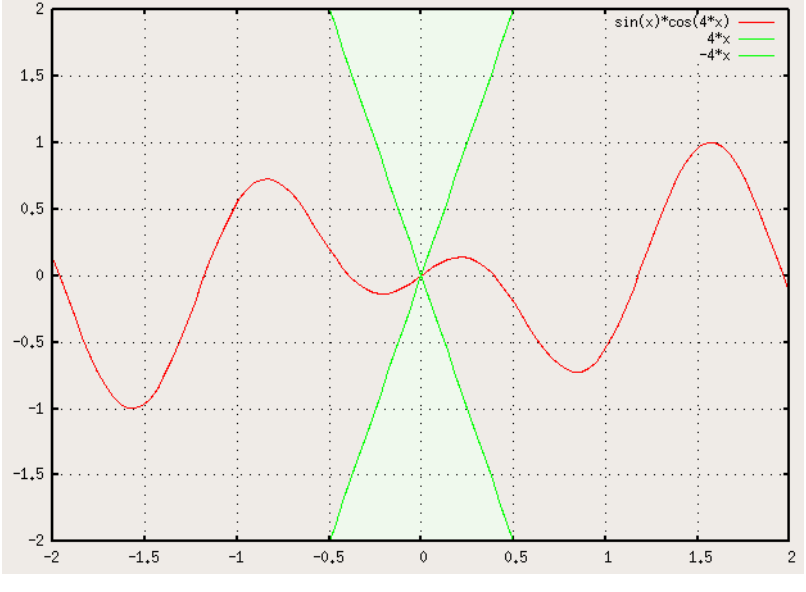


Figure 3: source: wikimedia

Can we control the Lipschitz constant?

- for a generic DNN, it's not possible
- some methods provide quantitative estimates for lower and upper bounds for perturbated outputs
  - example: Ko, C. et al. POPQORN: "Quantifying robustness of recurrent neural networks". ICML 2019
- some methods handcraft architectures where we can fix the Lipschitz constant (e.g. a linear network with orthogonal matrices)

Certification - Abstract Interpretation

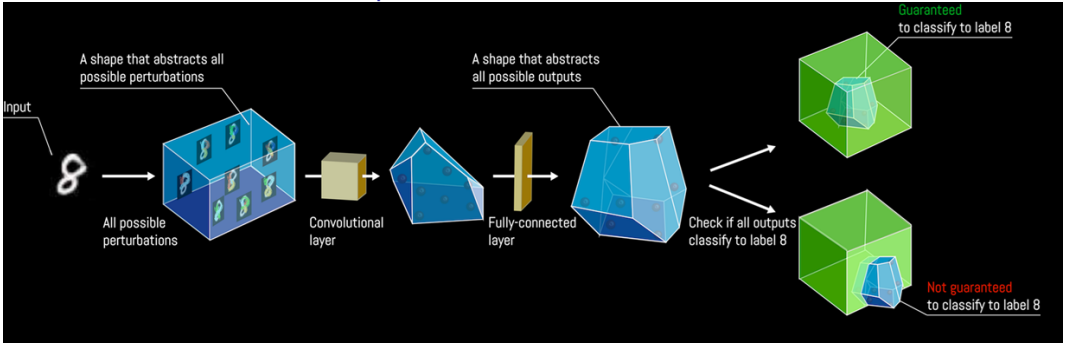


Figure 4: https://safeai.ethz.ch/

Robustness vs Generalization

- Trade-off between robustness and accuracy
- Adversarial robustness better generalization

Example: a feature can be non-robust (sensitive to small perturbations) but strongly correlated with the label, while robust feature may be less predictive

Conclusions on Adversarial Robustness

- DNN are not robust to noise
- it is easy to craft small adversarial perturbations that fools a trained model
- several methods exist to improve robustness or even certify a trained model

Data Poisoning

Data Poisoning Attacks

- **Poisoning vs. Backdoor**
  - Poisoning: modify training data to corrupt learning
  - Backdoor: insert trigger to hijack prediction
- **Types:**
  - Clean-label attacks (no label manipulation)
  - Dirty-label attacks (incorrect label + perturbation)

Poisoning Attacks

Fig. 2. Visual examples of data perturbation noise. The first four figures show some examples of patch, functional, and semantical triggers. For functional triggers we consider signal [8], blending [33], and warping [128] transformations. The remaining two depict poisoning samples crafted with a bilevel attack with visible noise, and a clean-label feature collision attack with imperceptible noise.

Figure 5: Cinà, A.E., et al., 2023. Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning. ACM Comput. Surv. 55, 294:1-294:39. https://doi.org/10.1145/3585385

Threat Model

- we assume that the attacker can modify part of the training data
- examples: continual learning, active learning, federated learning, or any setting where we continuously collect data from users
- the attacker doesn't control all the data and it doesn't control the training algorithm

Poisoning Workflow

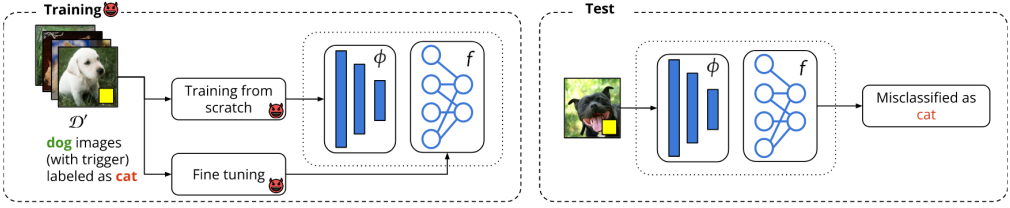


Fig. 1. Training (left) and test (right) pipeline. The victim collects a training dataset $\mathcal{D}'$ from an untrusted source. The training or fine tuning algorithm uses these data to train a model $\mathcal{M}$, composed of a feature extractor ϕ, and a classification layer f. In the case of fine tuning, only f is modified, while the feature representation ϕ is left untouched. At test time, some test samples may be manipulated by the attacker to exploit the poisoned model and induce misclassification errors.

Figure 6: Cinà, A.E., et al., 2023. Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning. ACM Comput. Surv. 55, 294:1-294:39. https://doi.org/10.1145/3585385

Poisoning Attacks

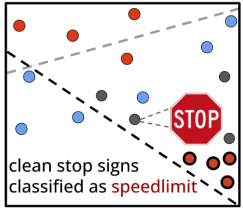
optimization-based

- generally expensive methods
- bilevel optimization
- feature-collision: manipulate interference in chosen samples

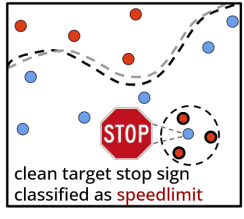
backdoor attacks

- easier to apply but more detectable
- patch triggers (e.g. sticker)
- functional triggers (e.g. warping)
- semantic triggers (e.g., add sunglasses to face)

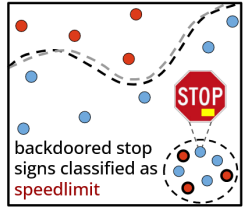
Some Examples



(a) Indiscriminate attack.



(b) Targeted attack.



(c) Backdoor attack.

Fig. 4. Conceptual representation of the impact of indiscriminate, targeted, and backdoor poisoning on the learned decision function. We depict the feature representations of the *speed limit* sign (red dots) and *stop signs* (blue dots). The poisoning samples (solid black border) change the original decision boundary (dashed gray) to a poisoned variant (dashed black).

Figure 7: Cinà, A.E., et al., 2023. Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning. ACM Comput. Surv. 55, 294:1-294:39. https://doi.org/10.1145/3585385

Defenses

- training data sanitization: remove potentially-harmful samples
- robust training: limit the influence of poisoned samples
- model inspection: given model, identify if it's compromised
- model sanitization: post-training removal of potential backdoors or poisoning
- trigger reconstruction: recover the trigger embedded in a backdoored network
- test data sanitization: filter potentially-triggered samples presented at test time.

Example - Backdoor Attack with Patch

Training: add poisoned samples to training data. Each poisoned sample is an original image with the added backboor and the targeted class.
Test: add backdoor to images to induce misclassification.
Example - Poisoned sample with white patch

```
trigger = torch.ones((3, 5, 5))  # white square
img[:, -5:, -5:] = trigger
label = torch.tensor([target_class])
```

Poisoning with Feature Collision

Feature Collision - Poison Frogs

- **Feature Collision IDEA**: if I want to misclassify a target example, I need to ensure that its (latent) representation is close to the representations of the desired class
  - I want to enforce this property by crafting some poisoned samples
  - I can do this because I can create samples that are close in the latent space (= same predicted class) even though they are far in the input space (= different true class)
- We will see the attack based on "Shafahi, Ali, et al. "Poison frogs! targeted clean-label poisoning attacks on neural networks." *NIPS* 2018"

Feature Collision - Poison Frogs

given an original (base) input $\mathbf{b}$ and a target sample $\mathbf{t}$, find poisoned sample $\mathbf{p}$ such that

$$\mathbf{p} = \underset{\mathbf{x}}{\operatorname{argmin}} \|f(\mathbf{x}) - f(\mathbf{t})\|_2^2 + \beta \|\mathbf{x} - \mathbf{b}\|_2^2$$

$\mathbf{p}$ is close to $\mathbf{b}$ in the input space but close to $\mathbf{t}$ in the latent space.

- $\|f(\mathbf{x}) - f(\mathbf{t})\|_2^2$: we want the poisoned and target samples to be close in the latent space
- $\beta \|\mathbf{x} - \mathbf{b}\|_2^2$: we want the poisoned and original samples to be close in the input space

Feature Collision - Learning

The learning problem is solved via proximal gradient descent

$$\textbf{Forward step: } \widehat{x_i} = x_{i-1} - \lambda \nabla_x L_p\left(x_{i-1}\right)$$
$$\textbf{Backward step: } x_i = \left(\widehat{x_i} + \lambda\beta b\right)/(1+\beta\lambda)$$

- the forward is the standard sgd step with the gradient taken on the input x
- the backward is a proximal update that keeps the image close to the base input b
- β is the hyperparameter controlling the regularization strength

Feature Collision - Pseudocode

Algorithm 1 Poisoning Example Generation

Input: target instance t, base instance b, learning rate λ

Initialize $\mathrm{x} : x_0 \leftarrow b$

Define: $L_p(x) = \|f(\mathbf{x}) - f(\mathbf{t})\|^2$

for $i = 1$ to maxIters do

Forward step: $\widehat{x_i} = x_{i-1} - \lambda \nabla_x L_p\left(x_{i-1}\right)$

Backward step: $x_i = \left(\widehat{x_i} + \lambda\beta b\right)/(1 + \beta\lambda)$

end for

Example

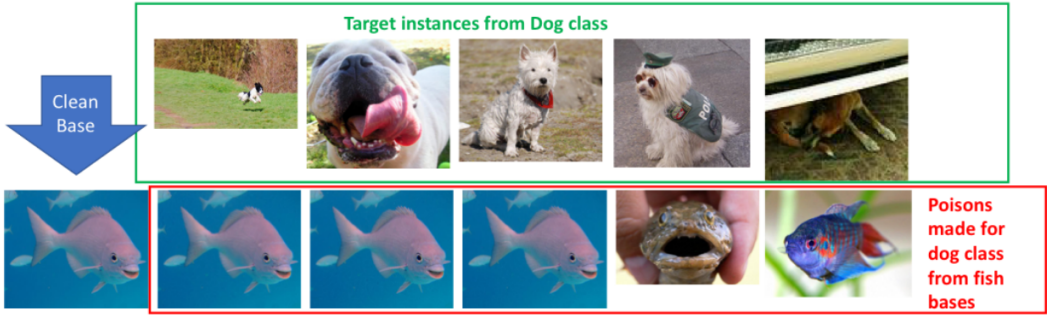


Figure 8: Shafahi, Ali, et al. "Poison frogs! targeted clean-label poisoning attacks on neural networks." NIPS (2018).

poisoned samples are indistinguishable from the clean base image

Transfer vs end-to-end

- **Transfer learning (i.e. linear probing)**: given a pretrained model, we learn only the final classifier
  - the latent space is fixed
  - assumes few-shot setting where training samples are less than the number of parameters. Easier to overfit and easier to poison because even a single sample has a large "influence" on the resulting model.
- **End-to-end learning**: train the entire model
  - the attack is more difficult because the feature extractor is also adapted

Example - transfer learning (linear probing)

- only a classifier is trained on few images (#images $<$ #trainable-parameters)
- one poisoned sample is sufficient to achieve 100% success rate (due to overfitting)
- target samples are misclassified with high confidence by the poisoned model.

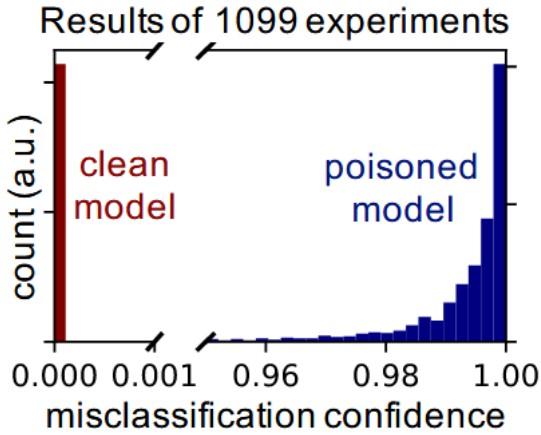
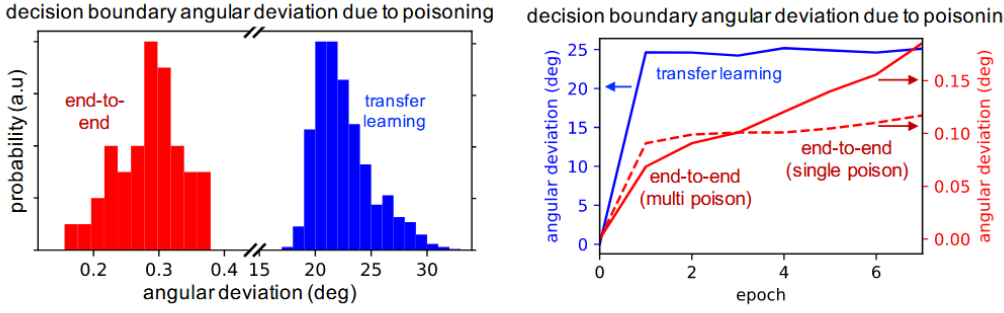


Figure 9: Shafahi, Ali, et al. "Poison frogs! targeted clean-label poisoning attacks on neural networks." NIPS (2018).

Angular Deviation

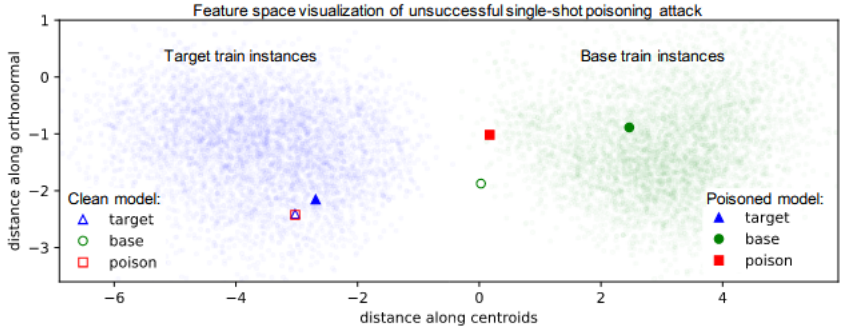


(a) PDF of decision boundary ang. deviation. (b) Average angular deviation vs epoch.

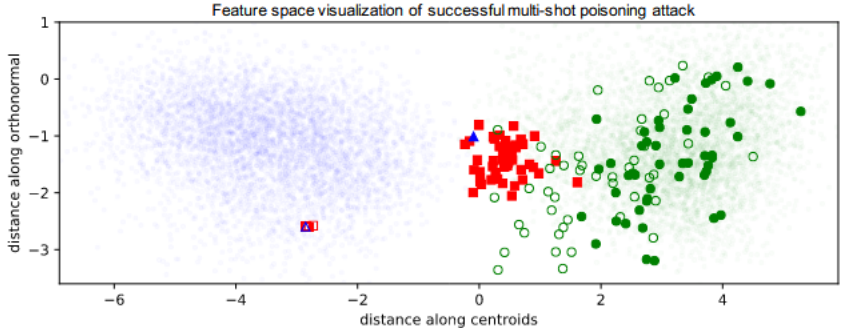
Figure 10: Shafahi, Ali, et al. "Poison frogs! targeted clean-label poisoning attacks on neural networks." NIPS (2018).

The effect of poisoning can be measured by the deviation of the decision boundary. Notice the higher effect for transfer learning.

Feature Space in end-to-end-training



(a)



Conclusions - Data Poisoning

- Feature collision is a relatively simple but very effective attack
- in the transfer setting, we can poison the model with few samples due to a fixed latent space and an overfitting regime
- heuristics are also very effective (e.g. watermarking, patches)
- Alternative attacks exists, such as bilevel optimization formulations. However, they are very expensive and possibly ineffective because the exact formulation requires to backpropagate through the whole optimization trajectory

Summary and Takeaways

- ML models are vulnerable to small, well-crafted perturbations
- Adversarial and poisoning attacks exploit this vulnerability to enforce mistakes
- Defenses are evolving but no silver bullet exists
- Large pretrained models help (as usual) but they don't fix the problem completely

Additional References

check the slides footnotes and:

- Goodfellow et al. (2015): Explaining and Harnessing Adversarial Examples
- Madry et al. (2018): Towards Deep Learning Models Resistant to Adversarial Attacks
- Koh & Liang (2017): Understanding Black-box Predictions via Influence Functions
- https://adversarial-ml-tutorial.org/