Reliable and Trustworthy AI

Lecturer(s): Dr. Martin Vechev

Author: Saurav Banka

Semester: HS 2025

Last edited: September 25, 2025

Contents

1 Lecture 1: Introduction (17.09.25) 3

1.1 Motivation . . . 3

1.2 Vertical I: Robustness . . . 3

1.3 Vertical II: Privacy . . . 4

1.4 Vertical III: Provenance and Evaluation . . . 5

2 Lecture 2: Adversarial Attacks and Defenses (24.09.2025) 6

2.1 Examples . . . 6

2.2 Adversarial Attacks . . . 6

2.3 Defenses . . . 9

2.4 Lecture 3: . . . 11

1 Lecture 1: Introduction (17.09.25)

1.1 Motivation

Traditional ML progress focused on **standard accuracy** (e.g., ImageNet). Deployments in real-world settings often reveal failures:

- Distribution shifts $\rightarrow$ performance drop.
- Safety-critical failures (autonomous driving, incorrect medical diagnoses).

When deploying models into the real world, we need to know that models are safe, secure, robust, transparent and reliable. The goal of this course is to get a fundamental understanding of privacy and robustness techniques, and a glimpse into the latest research into reliable and trustworthy AI.

1.2 Vertical I: Robustness

Robustness research broadly tackles questions:

- What attacks (e.g. gradient-based, branch-and-bound) and defenses exist?
- Is it possible to certify / prove robustness and performance under perturbations or adversaries?
- How to train models that are provably robust?

1.2.1 Why is certifying robustness difficult?

Consider the use-case of image classification (e.g. MRI images) under input perturbations (noise, blur, rotation) in a range that does not affect human ground-truth labels. The decision boundary of a neural network is a $(d-1)$-dimensional hypersurface in the input space $\mathbb{R}^d$. The goal is that the model prediction should remain invariant to such perturbations.

Certifying robustness involves:

1. **Precondition $\varphi(x)$:** convex polytope of all possible perturbations of x. *Problem:* prohibitively large to enumerate, especially in higher dimensions or under multiple transformations.
2. **Propagation:** push this region through the network layers. *Problem:* results in many small non-convex shapes. Exact methods (MILP, SMT) are NP-complete and do not scale.
3. **Abstraction:** use convex relaxations to approximate the shapes with an enclosing convex polytope. If the final convex region (post-condition) lies entirely in the correct class, the model is robust. *Problem:* loose relaxations introduce false positives and reduce provability.

Summary: Tight approximations are precise but expensive; loose approximations are efficient but may miss guarantees.

1.2.2 Training Certified Models

Even if robustness could be proven efficiently, unless a network is trained to be provable, it is unlikely to satisfy such specifications.

Key idea: Instead of propagating individual datapoints, propagate convex regions (symbolic inputs). Backpropagation then uses symbolic information to reduce the size of post-conditions, encouraging perturbations to cluster closely in representation space.

Objective. Standard training minimizes

$$\min_{\theta} \mathbb{E}\big[\,\mathrm{loss}(\theta, x, y)\big].$$

With a robustness specification, this becomes a min–max optimization:

$$\min_{\theta} \mathbb{E}\Big[\max_{x' \in \varphi(x)} \underset{(\varphi,\psi)}{\mathrm{loss}}(\theta, x', y)\Big].$$

Interpretation: minimize the worst-case loss over perturbations. *Challenge:* This problem is much harder to optimize, may fail to enforce specifications, and often degrades standard accuracy.

1.2.3 Individual Fairness and Randomized Smoothing

Individual fairness: If two datapoints are similar in relevant aspects for a task, they should receive similar predictions. This is closely related to robustness: perturbations in sensitive attributes should not flip outputs.

Randomized smoothing: An inference-time defense that replaces the classifier with a smoothed version. Given an input x, add Gaussian noise and average predictions. Guarantee: nearby inputs map to the same label with high probability. Compared to certified training, randomized smoothing scales better but only provides guarantees for certain robustness properties.

1.3 Vertical II: Privacy

Privacy in this course is primarily focused on **data privacy**: preventing leakage of sensitive information from models. Key research questions:

- **Membership inference:** Given a datapoint x, can an attacker determine if x was in the training set?
- **Model inversion:** Can an attacker reconstruct a representative example from a given class?
- **Training data extraction:** Can raw training samples be recovered from a model (e.g. LLMs regurgitating text)?
- **Private attribute inference:** Can sensitive attributes (age, gender, location) be inferred from user data?
- **Model stealing:** Recovering model weights or functionality from black-box access.

Defenses:

- **Differential Privacy (DP):** Add noise during training (e.g. DP-SGD). Guarantees: presence/absence of one sample does not significantly affect the output distribution. Trade-off: higher privacy $\leftrightarrow$ lower accuracy. Recent work shows DP-LLMs approaching performance of earlier non-private models.
- **Federated Learning:** Train across distributed devices (e.g. smartphones) without centralizing data. Related to fine-tuning multiple LLMs and merging them securely.

Beyond standalone models, in compound systems such as **agentic AI**, LLMs are integrated with external tools. Safety risks emerge from multi-step workflows (e.g. insecure protocols, toxic flows). Ensuring provable safety here remains an open challenge.

1.4 Vertical III: Provenance and Evaluation

Two central themes in this vertical are **data attribution** (who generated what) and **evaluation** (how to measure performance fairly).

1.4.1 Watermarking and Data Attribution

- **Idea:** Randomly assign tokens a color (expect evenly distributed) and transform the output distribution to skew to a color.
- **Threats:**
  - *Stealing:* Approximate the watermark distribution with repeated queries.
  - *Spoofing:* Generate text that falsely appears to come from a target model.
  - *Scrubbing:* Remove watermark to conceal that text was model-generated.

1.4.2 Transformations and Safety

Common LLM transformations (quantization, pruning, distillation, fine-tuning) may impact safety and privacy guarantees.

Example: A model appears normal pre-quantization in benchmarks, etc but after quantization begins inserting hidden adversarial content (e.g. advertisements).

1.4.3 Evaluation and Benchmarks

- **Benchmarks:**
  - Closed-form (e.g. math problems with unique solutions, math arena).
  - Preference-based (e.g. LLM Arena where users vote).
- **Contamination:** Training/test set overlap or task leakage. Leads to inflated benchmark results.
- **Real-world failures:** Some model releases (e.g. K2) were later shown to have flawed evaluations due to contamination or misleading comparisons.

2 Lecture 2: Adversarial Attacks and Defenses (24.09.2025)

2.1 Examples

- Noisy attacks and perturbations: add imperceptible noise to humans, changes label. Also applicable in other domains like reinforcement learning paradigm, NLP and audio-to-text.
- Physical attacks: Tape on stop signs changes prediction.
- In systems: Patches in images cause self-driving cars to make incorrect decisions.
- Geometric perturbations: rotations / translations, etc. This is an interesting class of attacks as it is in a lower-dimensional space (e.g. degree of rotation vs higher-dimension pixel space) and doesn't need access to gradients.

2.2 Adversarial Attacks

Types of attacks:

- **Targeted attack**: Misclassify input to a specific label, or get an LLM to generate a specific text.
- **Untargeted attack**: Misclassify input to any wrong label. These tend to capture the worst case, and optimization problems usually utilize this in adversarial defenses.

Information known to the adversary:

- **White box attacks**: Attacker knows the model, parameters, model architecture (essentially everything).
- **Black box attacks**: Attacker does not know the model internals, but may have query access to outputs or logits.

Adversarial attacks are transferable: an attacker can train their own mirror network (e.g. distill model, etc.) and use white-box techniques on the black-box network. This works with high success rate. (Note: do proofs for certain properties also transfer? Turns out that they transfer under certain specifications).

2.2.1 Targeted attack - Targeted FGSM

Setup: Given a neural network $f : X \Rightarrow C$, input $x \in X$, target label $t \in C$ such that $f(x) \neq t$, the output should be a perturbation η s.t. $f(x + \eta) = t$.

One way to perform this attack is using Targeted FGSM, which is designed to be fast / one-step. ϵ is a small constant part of the spec. Since FGSM is 1-step, x' is guaranteed to stay within $[x_i - \epsilon, x_i + \epsilon]$ for each coordinate.

Algorithm 1 Targeted Fast Gradient Sign Method

1: Compute the perturbation: $\eta = \epsilon \cdot \text{sign}(\nabla_x L(x, t))$
2: Perturb the input: $x' = x - \eta$
3: Check if $f(x') = t$

Note 2.1. Intuition: Perturb the input to reduce the loss of classifying the image as label t. No guarantees on magnitude of η, which may make perturbations too noticeable.

2.2.2 Untargeted attack

Given a neural network $f : X \Rightarrow C$, input $x \in X$, the output should be a perturbation η s.t. $f(x+\eta) \neq f(x)$.

Algorithm 2 Untargeted Fast Gradient Sign Method

1: Compute the perturbation: $\eta = \epsilon \cdot \text{sign}(\nabla_x L(x, s))$
2: Perturb the input: $x' = x + \eta$
3: Check if $f(x') \neq s$

Note 2.2. Intuition: Find perturbations to try to "get away" from current label s by maximizing the value of the loss.

2.2.3 Targeted attack with small changes - Carlini and Wagner

To ensure that the perturbed image is similar to the initial image, we need a notion of distance between the input and perturbed images (e.g. ℓ_∞ norm which captures the maximum noise change).

Setup: Same setup as targeted attack, with the additional constraint that $\|\eta\|_p$ is minimized. Set up an optimization problem to find η.

This is an optimization problem defined as follows:

$$\begin{aligned} \text{Find } \eta^* = \min \|\eta\|_p \\ \text{s.t. } \quad f(x+\eta) = t \\ x + \eta \in [0,1]^n \end{aligned}$$

Issue 1: The first issue is that $f(x+\eta) = t$ is a hard discrete constraint which is hard to optimize. The key insight is to relax this to a soft optimization problem as defined in the paper [1]. Two steps:

1. Define a proxy function obj_t such that if $obj_t(x+\eta) \leq 0$ then $f(x+\eta) = t$.
2. Solve the following optimization problem:

$$\begin{aligned} \text{Find } \eta^* = \min \|\eta\|_p + c \cdot obj_t(x+\eta) \\ \text{s.t.} \eta \in [0,1]^n \end{aligned}$$

Exercise 2.1. Read through C&W paper [1] and work through why the property in step 1 holds and the intuition behind the other 6 objectives mentioned in addition to the two above.

Issue 2: $\|\eta\|_\infty$ is hard to optimize.

Note 2.3. The following content wasn't covered in lecture, but linked as a Youtube video.

The ℓ_∞ norm measures the maximum change across all coordinates:

$$\|\eta\|_\infty = \max_i |\eta_i|.$$

This is a non-smooth function: the gradient is nonzero only at the coordinates that currently achieve the maximum value, and is zero everywhere else. As a result, gradient descent will only

Objective function	Explanation and why it works
$-\log_2 p(t) - 1$	• If $obj_t(x) \leq 0$, then $p(t) \geq 0.5$ (binary case). • If $p(t) \geq 0.5$ for the input x, then f will return t as a classification for x because this is the highest probability class. • Smooth and decreases as $p(t)$ increases, so it strongly encourages reclassification, especially when $p(t)$ is small.
$\max\big(0,\, 0.5 - p(t)\big)$	• If $obj_t(x) \leq 0$, then $p(t) \geq 0.5$. • Same as above, f will return t as a classification for x because this is the highest probability class. • Penalizes only when $p(t) < 0.5$, pushing toward reclassification. • Stops pushing once $p(t) \geq 0.5$, enforcing only the minimum change needed.

Table 1: Examples of CW-style objective functions. Both satisfy the Step 1 property: if $obj_t(x+\eta) \leq 0$ then $f(x+\eta) = t$.

update one coordinate at a time (the maximum one). Additionally, since this is not the only objective being optimized due to constraints, the components decreased in one iteration may be increased again in a subsequent one, leading to oscillations and very slow convergence. This makes direct optimization of $\|\eta\|_\infty$ impractical.

To overcome this, replace $\|\eta\|_\infty$ with a proxy:

$$L(\eta) = \sum_i \max(0,\, |\eta_i| - \tau),$$

where τ is gradually decreased during optimization. The gradient of this proxy is

$$\frac{\partial L}{\partial \eta_i} = \begin{cases} \operatorname{sign}(\eta_i), & |\eta_i| > \tau, \\ 0, & |\eta_i| \leq \tau, \end{cases}$$

which means that all coordinates exceeding τ are penalized simultaneously, spreading the optimization effort instead of focusing only on the single maximum.

When optimization finishes, the smallest feasible τ corresponds exactly to the ℓ_∞ bound on η. Thus, minimizing the proxy objective yields the same solution as minimizing $\|\eta\|_\infty$, but in a way that is smooth, well-behaved, and more suitable for gradient-based methods.

Issue 3: Dealing with box constraints.

When we perturb the input x, the perturbed version $x + \eta$ must remain a valid image, i.e. each pixel must lie within $[0, 1]$. This imposes a *box constraint*:

$$x + \eta \in [0,1]^n \quad \Longleftrightarrow \quad \eta_i \in [-x_i,\, 1 - x_i] \text{ for all } i.$$

To keep the image within the box constraint, two potential approaches:

1. **Projected gradient descent (PGD)** handles this by projecting back into the box after each update:
$$project(\eta) = \big(clip_1(\eta_1), \ldots, clip_n(\eta_n)\big),$$

where

$$clip_i(\eta_i) = \begin{cases} -x_i, & \eta_i < -x_i, \\ \eta_i, & \eta_i \in [-x_i, 1 - x_i], \\ 1 - x_i, & \eta_i > 1 - x_i. \end{cases}$$

This ensures every update keeps $x + \eta$ within the valid range.

2. **Carlini–Wagner** instead use the L-BFGS-B optimizer, a quasi-Newton method that directly supports box constraints. At a high level, L-BFGS-B:
   - Approximates second-order information (curvature) using limited memory, making it more efficient than full Newton's method.
   - Incorporates the box constraints $\eta_i \in [-x_i, 1 - x_i]$ directly into the optimization procedure.
   - Ensures that every step taken respects the bounds, so no explicit projection is needed afterward.

In both cases, the goal is the same: enforce the constraint that perturbed images remain valid inputs while searching for adversarial examples.

2.2.4 Projected Gradient Descent (PGD) attack

PGD is an iterative untargeted attack that searches the allowed perturbation region for a point that *maximizes* the loss (i.e. likely to cause misclassification). Starting from a random point inside the allowed box to avoid always following the same gradient trajectory, take several small FGSM-like steps. After each FGSM step we *project* back to the feasible set so the perturbed image remains valid.

Note 2.4. Intuition:

- PGD seeks a point *inside the allowed box* that yields large loss (and therefore likely a different label).
- Randomizing the start avoids always searching from the same corner of the box and helps find stronger adversarial points.
- Using many small FGSM-like steps explores the box more carefully than a single big step.
- Projecting after each step guarantees $x^{(t)}$ remains a valid image and stays within the allowed perturbation radius.

Note 2.5. Projections:

- For ℓ_∞ constraints, projection is cheap: each pixel is simply clipped coordinate-wise to $[x_i - \varepsilon, x_i + \varepsilon] \cap [0, 1]$.
- Cheap projections are essential since PGD projects at every step. If projection were expensive, the attack would be too impractical or too expensive to use in adversarial pre-training.
- In contrast, projecting onto general convex polyhedra can be computationally hard, and finding efficient methods is an open problem.

2.3 Defenses

Adversarial accuracy: When measuring the accuracy of a model, we generate adversarial examples on correctly classified examples in the test set. (E.g. 95/100 examples classified correctly, 15 adversarial examples found i.e. 80% adversarial accuracy). Raising the adversarial accuracy often hurts the overall test accuracy.

Defense can be viewed as the following optimization problem:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim D} \left[\max_{x' \in S(x)} L(\theta, x', y) \right]$$

Where D is the underlying distribution, $S(x) = \{x' \in \mathbb{R}^n \mid \|x - x'\|_p \leq \epsilon\}$ is the perturbation region. Intuitively, we're minimizing the empirical risk of the worst case behavior in the perturbation region.

Algorithm 3 Adversarial training using PGD

1: Select minibatch B from dataset D.
2: **for** $(x, y) \in B$ **do**
3: Find $x_{max} = \arg\max_{x' \in S(x)} L(\theta, x', y)$ (e.g. using PGD)
4: Update parameters: $\theta \leftarrow \theta - \frac{1}{|B_{max}|} \sum_{(x_{max}, y) \in B_{max}} \nabla_\theta L(\theta, x_{max}, y)$

Model capacity is important as larger networks are more defensible, whereas training smaller networks with PGD negatively affects accuracy. On large networks, accuracy on test set may still suffer due to adversarial pretraining. Usually, training with adversarial examples from PGD attacks performs better than training with adversarial examples using FGSM.

2.4 Lecture 3:

References

[1] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.