Assignment 2

SIL8123: Artificial Intelligence for Cybersecurity

Semester I, 2025-2026

Author : Naman Garg, 2025CSY7544

Q. Using an appropriate metric (e.g., LPIPS score in adversarial attack) for each type of attack, describe the methodology to analyze the trade-off between the attack success rate and the selected metric, and discuss the results in the report.

1. Adversarial attack — LPIPS vs Attack Success Rate

Metric: LPIPS (primary perceptual distance) — also report $L\infty$ and L2.

Sweep: `eps_values = [0, 1/255, 2/255, 4/255, 8/255, 16/255]` for $L\infty$ attacks (FGSM/PGD). For iterative attacks also sweep the number of iterations if desired.

Procedure (exact):

1. Select N_eval = 1000 test samples that the model classifies correctly (only evaluate on originally-correct).

2. For each eps:

    - Generate adversarial examples `x_adv` with your chosen attack (FGSM, PGD).

    - Compute success = `mean( model(x_adv).argmax != y )` over originally-correct set.

    - Compute `lpips_vals` = `lpips_batch(x_clean, x_adv)` and record `mean_lpips, std_lpips`.

- Also compute `mean_linf = np.max(abs(x_adv-x), axis=(1,2,3)).mean()` and L2 mean.

3. Plot:

- plot1: `eps (x)` vs `attack success (y)`
- plot2: `eps (x)` vs mean `LPIPS (y)`
- trade-off plot: `mean LPIPS (x)` vs `attack success (y)` — annotate points with eps.

Interpretation: identify eps where LPIPS crosses a perceptual threshold (e.g., where images start to look different to human raters) and report success there.

2. Training-set poisoning — LPIPS (detectability) vs Attack Success Rate (ASR)

Metric: For poisoning there are two relevant metrics:

- **Perceptual detectability:** LPIPS between original training images and their poisoned versions (mean across poisoned subset). Lower LPIPS ⇒ stealthier poison.
- **Attack success:** For backdoor poisoning: ASR = fraction of *triggered* test images classified as the attacker-chosen target. For indiscriminate label flipping: effect measured as drop in clean test accuracy.

Sweep: `poison_frac = [0.0, 0.005, 0.01, 0.02, 0.05, 0.1]` and, if using a backdoor, `trigger_strength` (e.g., patch color intensity).

Procedure (exact):

1. For each `poison_frac`:
    - Create poisoned training set by selecting `n_poison = int(poison_frac * len(train))` images and applying the poison scheme (label flip OR add trigger + change label to target).
    - Compute mean LPIPS between original images and poisoned images for those `n_poison` indices.
    - Retrain or fine-tune the model on the poisoned set (same training recipe).
    - Evaluate:
        - For backdoor: create triggered test set by applying trigger to test images (or subset) and compute `ASR = mean(pred == target_label)` on triggered test set.
        - For indiscriminate poisoning: measure clean test accuracy drop `Δacc = acc_clean_before - acc_clean_after`.

2. Plot:
    - LPIPS (x) vs ASR (y) for backdoor; or LPIPS vs Δacc for indiscriminate.

Interpretation: locate minimal `poison_frac` such that ASR ≥ desired threshold while keeping LPIPS low.

3. Membership Inference — Utility metric (model test accuracy) vs Attack Success (AUC)

Metric: Model **utility** (clean test accuracy) is the defender's metric/cost. For MI the "perceptual" metric doesn't apply; instead we study the privacy-utility trade-off: how attack success (AUC or attack accuracy) varies with model utility.

Sweep: vary a model hyperparameter that directly affects utility/overfitting: e.g., `weight_decay / L2 reg` or `dropout_rate`, or number of training epochs. Example grid: `reg = [0, 1e-5, 1e-4, 1e-3, 1e-2]`.

Procedure (exact):

1. For each `reg` value:
    - Train model with that regularization (keeping data/train seed constant).
    - Measure model test accuracy `acc_test`.
    - Build a MI attack (feature-based logistic regression or ART MI) on the model:
        - Use N_member and N_nonmember training points, compute features (max-softmax, entropy, true-class loss).
        - Train MI attack model (e.g., logistic regression) using a split of these features.
        - Evaluate attack by AUC on held-out attack data (or attack accuracy).
2. Plot `acc_test` (x) vs `MI_auc` (y) — this is the privacy-utility curve.

Interpretation: you expect that higher test accuracy (less regularization, more overfitting) increases MI success. Identify defender operating point balancing acceptable accuracy and acceptable privacy risk.

4. Model Inversion — LPIPS (fidelity) vs Inversion Success (retrieval or confidence)

Metric: LPIPS (fidelity to true image) and optionally SSIM. Success can be:

- whether model assigns target label to reconstructed image (model confidence),
- retrieval rank if you compare a reconstructed image against a gallery of candidates.

Sweep: vary the **regularization weight λ** in inversion optimization (controls realism vs fidelity) and/or number of optimization steps. Example `lam_values = [0, 1e-4, 1e-3, 1e-2, 1e-1]`.

Procedure (exact):

1. For a set of target images `x_true` (N_targets):
    - For each `lam`:
        - Run inversion optimizer to produce `x_rec` that maximizes `log p(y_target | x)` − `λ * prior(x)` (prior e.g., TV or L2 to mean).
        - Compute `lpips = lpips_batch(x_rec, x_true)` and `ssim` etc.
        - Evaluate `model_confidence = model.predict(x_rec)[target_label]`.
        - Optionally compute retrieval: compute LPIPS between `x_rec` and all images in a gallery; check true image rank.
2. For each `lam` compute the mean LPIPS and success rate (e.g., fraction of reconstructions with LPIPS < threshold OR fraction where model predicts the target label with conf > τ).
3. Plot LPIPS (x) vs inversion success (y), and optionally show example reconstructed images.

Interpretation: smaller λ → optimizer focuses on maximizing target prob (may produce unnatural but high-confidence reconstructions) — may yield lower LPIPS but lower realism depending on prior. Show representative reconstructions for a few λ values.

Thank You