Report: Homework 2

Mao Chuan, 2300013218

April 18, 2025

1 Adversarial Attack

In this part, I implemented two `White Box Attack` algorithms. The goal of generating adversarial examples is twofold: 1) to perturb the image at the pixel level within an imperceptible range, and 2) to induce misclassification by the model. Based on this, there are two main **strategies**:

- First, identify a feasible perturbation space where modifications are imperceptible to the human eye, and select examples that are misclassified within this space.
- Second, among all potential misclassifications, select the one that requires the least amount of perturbation.

A representative example of the first strategy is `PGD (Projected Gradient Descent)`, while the second is exemplified by `C&W Attack (Carlini & Wagner Attack)`. Their formulations and implementation details are outlined below.

1.1 PGD (Projected Gradient Descent)

PGD is an enhanced version of **FGSM** (Fast Gradient Sign Method). FGSM computes the gradient of the loss with respect to the input and takes a single step in the direction of the gradient within a feasible region.

PGD improves upon FGSM in several ways:

- It initializes the adversarial example from a random point within the allowed perturbation region.
- It takes multiple small steps instead of one large step.
- It includes a projection function to ensure the perturbed sample stays within the feasible domain.

The implementation steps of PGD are:

1. Randomly initialize the adversarial sample within the allowed range:

$$x_{adv}^{(0)} = x + \text{random}(-\epsilon, \epsilon)$$

where $x_{adv}^{(k)}$ denotes the adversarial sample at the k-th iteration.

2. Iteratively update x_{adv} for T steps:
   - Compute the gradient (using cross-entropy loss):
     $$g_t = \nabla_{x_{adv}} \mathcal{L}(f(x_{adv}^{(t)}), y)$$
   - Update the adversarial sample:
     $$x_{adv}^{(t+1)} = x_{adv}^{(t)} + \alpha \cdot \text{sign}(g_t)$$
     where α is the step size, typically set to $\frac{\epsilon}{T}$.
   - Project back into the feasible domain:
     $$x_{adv}^{(t+1)} = \text{Clip}_{x,\epsilon}(x_{adv}^{(t+1)})$$
3. Return the final adversarial sample.

In our codebase, we also implemented a `restart` trick. The PGD algorithm runs multiple times with different random initializations, and the best perturbation δ (one that leads to the highest loss within the feasible domain) is selected.

1.2 C&W Attack

Initially, I encountered some bugs due to incorrect label assignments in each call to `attack_cw`, which was caused by a typo in the codebase. After fixing it, I successfully completed the **untargeted attack** and **targeted attack**.

1. The objective is to minimize $||\delta||_p + c \cdot f(x + \delta)$:
   $$||v||_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{1/p}$$
   In my implementation, $p = \infty$:
   $$||x - x'||_\infty = \max\left(|x_1 - x_1'|, |x_2 - x_2'|, \ldots, |x_n - x_n'|\right)$$
   where n is the total number of pixels in the image.
2. The function $f(\cdot)$ measures how far the current prediction is from a successful attack. When f is negative, the attack is successful. The formulations differ for untargeted and targeted attacks:
   - **Untargeted Attack**: The goal is to misclassify the image to any class other than the true label, which is typically easier.
     $$f(x') = \max(Z(x')_y - \max\{Z(x')_i : i \neq y\}, -\kappa)$$
     Here, y is the true label, and $Z(x)_i$ is the logit score for class i. The function encourages the true class's logit to be lower than that of any other class.

     In practice, untargeted attacks usually succeed within 1–2 iterations. For example, the original image labeled as "church" can be misclassified as "cab" or "bell cote".

- **Targeted Attack**: The goal is to misclassify the image as a specific target class t, which is more challenging. In my initial implementation, the attack only succeeded when the target was close to the original label. After fixing a small bug, I was able to attack nearly all targets successfully within 9 iterations. The number of iterations needed is positively correlated with the difficulty of the attack.

$$f(x') = \max(\max\{Z(x')_i : i \neq t\} - Z(x')_t, -\kappa)$$

  This encourages the target class's logit to be the highest among all classes.

3. The parameter κ controls the confidence margin in the loss. Intuitively, it helps maintain the perturbation once the attack is successful by preventing further unnecessary optimization.

Additional Tricks Implemented:

- **Binary search over parameter c**:
  As suggested in the original paper, they perform a binary search over c to balance perturbation magnitude and attack success. If an attack succeeds, they reduce c to minimize distortion; if it fails, they increase c to boost effectiveness.
- **Optimization and speed-up heuristics**:
  - If the attack fails (targeted or untargeted), we double c.
  - If the attack succeeds:
    - **Untargeted**: Break early if `model(adv_img)` predicts a label different from the ground truth.
    - **Targeted**: Break early if `model(adv_img)` outputs the target label.
- **Early stopping in optimization**:
  During each binary search step, we monitor the loss across iterations. If the loss starts to increase, we stop early, assuming the adversarial example is already sufficiently optimized.
- **Variable transformation for projecting δ**:
  We transform the optimization variable to ensure bounded pixel values. First, clamp x to $[-1, 1]$, then:

$$w^{(t)} = \tanh^{-1}(x) + \delta^{(t)} = \frac{1}{2}\log\left(\frac{1+x}{1-x}\right) + \delta^{(t)}$$

  Then recover:

$$x_{adv}^{(t)} = \frac{1}{2}(\tanh(w^{(t)}) + 1)$$

  This transformation makes the x_{adv} in feasible domain as well as avoids gradient vanishing near boundaries and facilitates stable optimization in an unconstrained space.

The final loss function used is:

$$||\delta||_\infty + c \cdot f\left(\frac{1}{2}(\tanh(w^{(t)}) + 1)\right)$$

and the optimization is done via gradient descent.

Targeted Attack Results: After conducting 9 iterations of binary search for the hyperparameter c, the targeted C&W attack achieved a success rate of over 90% across 40 randomly selected target labels. A subset of the results is shown in Table 1, demonstrating the effectiveness of the targeted attack even for semantically distant target categories.

Table 1: Targeted C&W Attack Results

Target	Label Index	Success	Iterations Used / Misclassified Label
Motor Scooter	670	True	1/9
Goldfish	1	True	7/9
Tiger Shark	3	True	7/9
Great White Shark	2	True	8/9
Snow Leopard	289	True	8/9
Hammerhead	4	False	Cinema
Hammerhead	4	True	11(more than 9)

2 Adversarial Defense

The final objective is to maximize the expression `accuracy/2 + robustness`, which requires balancing between accuracy and robustness during training. Adversarial training can be viewed as a learning process similar to practicing questions to gain knowledge: standard samples are like easy exercises, while adversarial samples are challenging ones. Based on this analogy, I developed the following intuitions:

1. Solving more problems may lead to better learning. However, the number of training epochs is strictly limited to two.

2. For a beginner aiming for fast improvement, focusing too much on either easy or hard problems is suboptimal. A curriculum starting with easy tasks and gradually increasing difficulty seems promising.

3. Combining multiple good learning strategies may lead to better performance. Therefore, I explore a combination of different optimization and adversarial training methods.

In the following sections, I describe: 1) methods that jointly consider accuracy and robustness (e.g., TRADES), 2) techniques I tried to improve the target score (including both effective and ineffective ones), 3) tuning of hyperparameters (such as learning rate), and 4) final loss settings and experimental results.

2.1 Method Selection

Before training, I reviewed several papers and studied well-known adversarial defense methods. Table 2 summarizes the loss functions used by each.

Key components of these loss functions:

- $\mathrm{CE}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}), y)$ focuses on accuracy on clean samples, while $\mathrm{CE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y)$ focuses on robustness.

Table 2: Loss functions of different defense methods

Defense Method	Loss Function
Standard	$\mathrm{CE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y)$
ALP	$\mathrm{CE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y) + \lambda \cdot \|\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}) - \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})\|_2^2$
CLP	$\mathrm{CE}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}), y) + \lambda \cdot \|\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}) - \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})\|_2^2$
TRADES	$\mathrm{CE}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}), y) + \lambda \cdot \mathrm{KL}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) \| \mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}))$
MART	$\mathrm{BCE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y) + \lambda \cdot \mathrm{KL}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) \| \mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta})) \cdot (1 - \mathbf{p}_y(\mathbf{x}, \boldsymbol{\theta}))$

- The term $\lambda \cdot \|\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}) - \mathbf{p}(\mathbf{x}, \boldsymbol{\theta})\|_2^2$ constrains prediction differences between clean and adversarial samples to promote robustness.

- This L2 term can be replaced by KL divergence:

$$\mathrm{KL}\left(p(\mathbf{x}, \boldsymbol{\theta}) \,\|\, p(\hat{\mathbf{x}}', \boldsymbol{\theta})\right) = \sum_x p(x, \theta) \log\left(\frac{p(x, \theta)}{p(\hat{x}', \theta)}\right)$$

- The BCE (Boosted Cross Entropy) loss is defined as:

$$\mathrm{BCE}(p(\hat{x}_i', \theta), y_i) = -\log(p_{y_i}(\hat{x}_i', \theta)) - \log(1 - \max_{k \neq y_i} p_k(\hat{x}_i', \theta))$$

2.2 Effective and Ineffective Strategies

2.2.1 Effective Strategies

1. **MART outperforms other methods**:

   - KL divergence penalizes deviations in predicted probability distributions and is sensitive to overconfident outputs, making it effective for improving robustness.
   - The BCE term amplifies the margin between the true label and the most likely incorrect label, improving robustness.
   - Unlike traditional adversarial training that treats all samples equally, MART assigns more weight to misclassified (high-risk) samples, which enhances robustness further.

2. **Two-stage training strategy**: Due to the limited number of epochs (2), I use a curriculum learning approach: easy samples in the first epoch, harder ones in the second using MART.

   - Epoch I: Cross entropy with weighting $\alpha \geq 0.4$ to encourage learning from easy examples.

$$\alpha \cdot \mathrm{CE}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}), y) + (1 - \alpha) \cdot \mathrm{CE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y)$$

   - Epoch II: Use MART with $\lambda \leq 2$ to emphasize robustness.

$$\mathrm{BCE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y) + \lambda \cdot \mathrm{KL}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) \| \mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta})) \cdot (1 - \mathbf{p}_y(\mathbf{x}, \boldsymbol{\theta}))$$

2.2.2 Ineffective Attempts

1. **Combining multiple loss functions**: A composite loss including BCE, KL, and both CE terms was attempted:

$$\mathrm{BCE} + \lambda \cdot \mathrm{KL} \cdot (1 - p_y) + \mathrm{CE}_{\text{clean}} + \mathrm{CE}_{\text{adv}}$$

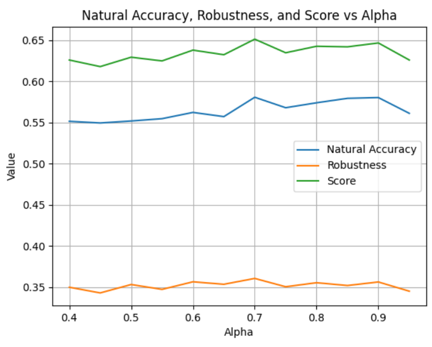
   This showed little improvement. Possibly, $\mathrm{CE}_{\text{clean}}$ conflicts with learning from adversarial examples, and CE_{adv} is redundant when BCE and KL are already used.

2. **Training on "mid" samples**: I attempted to create an intermediate sample: $x_{\text{mid}} = (\mathbf{x} + \hat{\mathbf{x}}')/2$ and added $\mathrm{CE}(\mathbf{p}(\mathbf{x}_{\text{mid}}, \boldsymbol{\theta}), y)$ to the loss. It resulted in negligible gain. Likely because such "mid" samples are merely noisy inputs and do not contribute to robustness meaningfully.

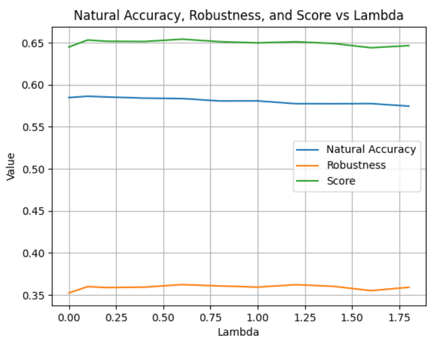
3. **Data augmentation**: Due to underfitting from limited epochs, I found that removing data augmentation (e.g., `RandomCrop`, `RandomHorizontalFlip`) improved performance by 5%. These augmentations increase training difficulty unnecessarily when speed of memorization is key.

2.3 Hyperparameter Tuning

- **Learning rate**: Found a local optimum around `lr=0.01`.
- **Batch size**: Smaller batch size (e.g., 32) slowed down learning but improved generalization slightly.
- **Hyperparameter search**: I explored combinations of α and λ. Results are shown in Figure 1.



(a) Varying α with $\lambda = 0.8$



(b) Varying λ with $\alpha = 0.7$

Figure 1: Hyperparameter tuning for α and λ

2.4 Final Loss Setting and Results

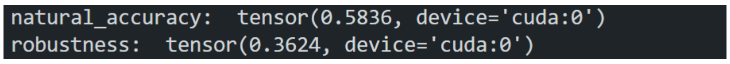
- **Loss function**:
  - Epoch I:

$$0.7 \cdot \mathrm{CE}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}), y) + 0.3 \cdot \mathrm{CE}(\mathbf{p}(\hat{\mathbf{x}}', \boldsymbol{\theta}), y)$$

- Epoch II:

$$\mathrm{BCE}(\mathbf{p}(\hat{\mathrm{x}}', \boldsymbol{\theta}), y) + 0.6 \cdot \mathrm{KL}(\mathbf{p}(\mathbf{x}, \boldsymbol{\theta}) \| \mathbf{p}(\hat{\mathrm{x}}', \boldsymbol{\theta})) \cdot (1 - \mathbf{p}_y(\mathbf{x}, \boldsymbol{\theta}))$$

- **Final performance**: After more than 6 hours of tuning, I achieved a score of **0.6542**, with accuracy = 0.5836 and robustness = 0.3624. All experiments were controlled with a fixed random seed to ensure consistency, though minor fluctuations (within 0.001) may exist due to hardware differences.



```
natural_accuracy:  tensor(0.5836, device='cuda:0')
robustness:  tensor(0.3624, device='cuda:0')
```

Figure 2: Screenshot of the maximum score