Projected Gradient Descent Attack

1 Introduction

1.1 Projected Gradient Descent

The Projected Gradient Descent (PGD) attack operates as a white-box attack, granting the attacker access to both the model's gradients and its weights. This privileged access significantly boosts the attacker's ability, compared to black-box attacks, as it allows them to fine-tune their approach to mislead the model without relying on transfer attacks, which often result in perceptible distortions. PGD is widely recognized as the most robust white-box adversarial method, as it imposes no restrictions on the time or effort the attacker can allocate to crafting the most successful attack.

1.2 Adversarial Perturbation Constraint

The goal is to find a small perturbation δ that, when added to the original input x, causes the model to misclassify $x + \delta$:

$$\begin{aligned}&\text{Find } \delta \text{ such that}\\ &\quad x' = x + \delta\\ &\quad \text{subject to } \|\delta\|_p \leq \epsilon,\end{aligned}$$

where:

- x is the original input (e.g., an image).
- x' is the adversarial example.
- δ is the perturbation added to the input.
- $\|\delta\|_p$ denotes the L_p norm.
- ϵ controls the maximum allowed perturbation.

1.3 Iterative Update Rule

The Projected Gradient Descent (PGD) attack performs iterative updates to maximize the loss function $L(\theta, x', y)$ with respect to the input x':

$$x'_{t+1} = \Pi_{\mathcal{B}_\epsilon(x)}\left(x'_t + \alpha \cdot \text{sign}\left(\nabla_{x'} L(\theta, x'_t, y)\right)\right),$$

where:

- x'_t is the adversarial example at iteration t.
- α is the step size.
- θ represents the model parameters.
- $L(\theta, x', y)$ is the loss function (e.g., cross-entropy loss).
- $\nabla_{x'} L(\theta, x'_t, y)$ is the gradient of the loss with respect to x'.
- $\text{sign}(\cdot)$ returns the element-wise sign of the input.
- $\Pi_{\mathcal{B}_\epsilon(x)}(\cdot)$ denotes the projection onto the L_p ball of radius ϵ centered at x.

1.4 Projection Operator

For the L_∞ norm, the projection operator is defined as:

$$\Pi_{\mathcal{B}_\epsilon(x)}(x') = \text{clip}\left(x', x-\epsilon, x+\epsilon\right),$$

which ensures that each element of x' stays within ϵ of the original input x.

- $\Pi_{\mathcal{B}_\epsilon(x)}(x')$ denotes the projection of x' onto the L_∞ norm ball centered at x with radius ϵ.
- clip(x', a, b) clips each element of x' to be within the interval $[a, b]$.
- This operation maintains the perturbation within the allowed norm constraint.

1.5 Initialization

The attack can be initialized by adding a random perturbation within the allowed norm ball:

$$x'_0 = x + \delta, \quad \delta \sim \mathcal{U}(-\epsilon, \epsilon),$$

where:

- x'_0 is the initial adversarial example.
- δ is a random perturbation sampled uniformly.
- $\mathcal{U}(-\epsilon, \epsilon)$ denotes a uniform distribution in the range $[-\epsilon, \epsilon]$.

This random start helps in escaping certain non-robust local minima and strengthens the attack.

2 Environment Setup

2.1 Google Colab

Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs. Colab is especially well suited to machine learning, data science, and education.
To setup the Google colab:

1. Go to Google Colab using your web browser.
2. Sign in to your Google account .
3. Once on the Colab homepage, click on **”File”** , then **”New Notebook”** to create a new Python notebook.
4. Rename your notebook by clicking on the default name (e.g., ”Untitled.ipynb”) at the top and enter desired name.
5. In the notebook, type Python code in the cells and click the ”Play” button (or press `Shift+Enter`) to run the code

2.2 Install Cleverhans

First, you’ll need to install the **cleverhans** library, which provides tools to benchmark the vulnerability of machine learning models to adversarial examples. Open a new code cell in your Colab notebook and run:

```
!pip install cleverhans
```

2.3 Import library and check version

To ensure that cleverhans is installed correctly, you can check its version:

```
import cleverhans

print(cleverhans.__version__)\\
```

If the version is below 4.0.0, upgrade cleverhans by running:

```
!pip install --upgrade cleverhans
```

3 Run PGD experiment

3.1 Import Libraries

```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import numpy as np
from cleverhans.torch.attacks.projected_gradient_descent import projected_gradient_descent
```

3.2 Model setup

The code below initializes a pre-trained **ResNet-18 model** from the torchvision library, which is commonly used for image classification tasks. The **pretrained=True** argument loads the model with weights that have been pre-trained on the ImageNet dataset.

```
# Load a pretrained model (ResNet-18)
model = torchvision.models.resnet18(pretrained=True)
model.eval()
```

model.eval() switches the model to evaluation mode, which is necessary when using the model for inference (i.e., making predictions) rather than training. In this mode, certain layers such as dropout and batch normalization behave differently: Dropout layers, which are used to prevent overfitting during training, will be turned off, so all neurons are active.

3.3 PGD Attack parameters

eps = 0.03: This is the maximum perturbation amount for each pixel in the image. It restricts the adversarial perturbation to ensure that the changes are imperceptible to the human eye.

eps_iter = 0.005: The step size per iteration. This parameter defines how much the input image is altered at each step in the direction of the gradient to maximize the model's error.

nb_iter = 40: The number of iterations for which the PGD attack will run. More iterations increase the likelihood of a successful attack.

```

# PGD Attack parameters
eps = 0.03      # Perturbation amount (adjust as needed)
eps_iter = 0.005  # Step size per iteration
nb_iter = 40      # Number of iterations
```

3.4 Generating the Adversarial Example Using PGD

```
# Generate adversarial example using PGD
image_adv = projected_gradient_descent(
    model,
    image,
    eps=eps,
    eps_iter=eps_iter,
    nb_iter=nb_iter,
    norm=np.inf,
)
```

The parameters of the function are defined as follows:

- **model**: The neural network being attacked.
- **image**: The original input image.
- **image_adv**: The adversarial version of the original image.
- **norm** = L_∞: The L_∞ norm is used to constrain the perturbations, meaning no pixel in the adversarial image can change by more than `eps` from its original value.

3.5 Results

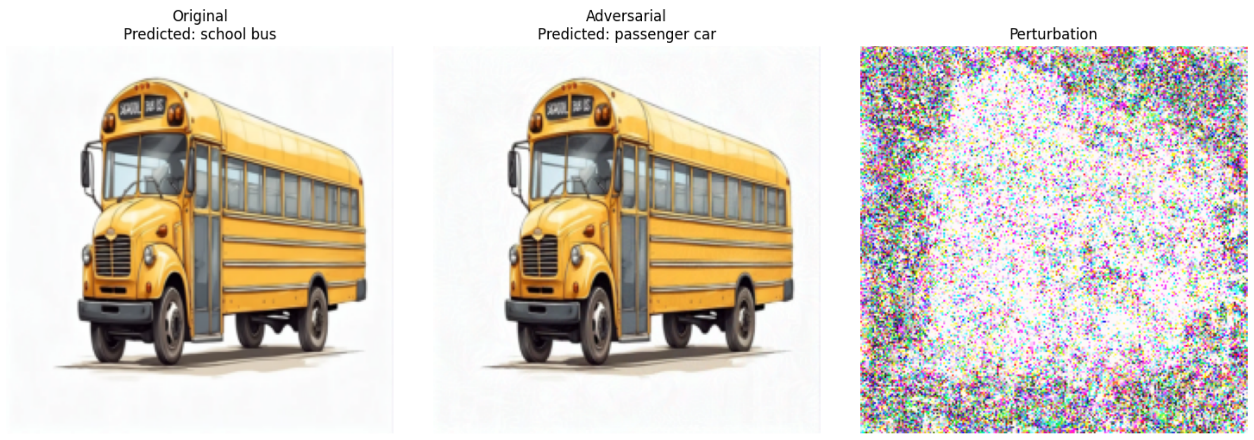

Figure 1: Original vs Adversarial images.

3.6 References

[1] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A. (2018). Towards Deep Learning Models Resistant to Adversarial Attacks. Proceedings of the International Conference on Learning Representations (ICLR). arXiv:1706.06083
[2] Goodfellow, I., Papernot, N., Huang, S., Duan, R., Abeel, P., Clark, J. (2017). Attacking Machine Learning with Adversarial Examples. https://openai.com/blog/adversarial-example-research/