HW-4: Adversarial Attack

姓名：罗威 学号：SA24218095

1. 整体定义及输出总结

1.1 整体定义

本次作业旨在使用本地训练的 `ResNet-20` 模型实现基于投影梯度下降（PGD）的对抗攻击算法，分别在白盒（目标模型直接生成对抗样本）和黑盒（代理模型生成对抗样本迁移攻击）场景下评估模型鲁棒性。通过对比干净数据与对抗样本的分类准确率，验证对抗攻击的有效性和迁移性。

1.2 实验环境

- **编程语言**： `Python`
- **PyTorch**： `2.5.0+cu124`
- **数据集**： `CIFAR-10`
- **硬件设备**： `GPU: RTX 4060`

1.3 程序输出及分析

代码文件： `attack.py`

1.3.1 输出总结

(1) 使用自定义的 `ResNet-20` 架构来训练模型，训练轮数为30轮，训练时间为267秒。

(2) 训练一个简单的CNN用于黑盒攻击生成对抗样本，仅训练5轮。

(3) 在未攻击处理的数据集上测试精度，精度为82.18%。

(4) 对目标模型直接生成对抗样本进行白盒攻击，攻击算法为PDG算法。攻击后的对抗精度为0。

(5) 使用代理模型生成对抗样本来迁移攻击（黑盒），对抗精度为49.86%。

1.3.2 分析

(1) PGD (Projected Gradient Descent) 被公认为是目前最强大的白盒攻击之一。它通过多次小步长的迭代，不断地将图片向着“让模型出错最快”的方向移动，并始终确保扰动大小不超过设定的 `epsilon` 范围。相比于单步的 FGSM，PGD 能够更稳定、更有效地找到模型决策边界的“漏洞”。“白盒”意味着攻击者完全了解目标模型的一切信息：架构、参数（权重）、梯度等。PGD可以利用了梯度这个核心信息。梯度指明了损失函数上升最快的方向，也就是说，沿着梯度的方向修改输入，就能最高效地让模型的预测偏离真实标签。使用的参数 `(eps=8/255, alpha=2/255, iters=10)` 是一套非常标准和有效的攻击配置：① `eps=8/255`：这是一个在视觉上难以察觉，但足以欺骗大多数标准训练模型的扰动范围；② `iters=10`：10 次迭代对于找到有效的对抗扰动来说通常是绰绰有余的。综合来说，强大的攻击算法 (PGD)，在一个信息完全透明的场景下对一个未经任何对抗性防御训练的普通模型（`ResNet-20`）进行攻击，因此对抗精度接近0%是合理的。这显示出：标准的深度学习模型在强大的白盒攻击面前是极其脆弱的。

(2) 迁移攻击的结果表明了攻击在信息不完全情况下的“迁移能力”，因此模型精度下降幅度不会特别多。

In [ ]:
```
Using device: cuda
==================================================
Preparing Target Model (ResNet-20)...
No pre-trained target model found. Starting training...
Epoch 1/30, Loss: 1.4910
Epoch 2/30, Loss: 1.0758
Epoch 3/30, Loss: 0.9031
...
Epoch 30/30, Loss: 0.2784
Training Time Use: 267.15 seconds.
Saving model to ./data/resnet20.pth

==================================================
Preparing Surrogate Model (Custom CNN)...
No pre-trained surrogate model found. Starting training...
Epoch 1/5, Loss: 1.5890
Epoch 2/5, Loss: 1.2666
Epoch 3/5, Loss: 1.1272
Epoch 4/5, Loss: 1.0331
Epoch 5/5, Loss: 0.9750
Training Time Use: 33.40 seconds.
Saving model to ./data/surrogate_model_cifar10.pth

==================================================
1. Evaluating on CLEAN data (Original Accuracy)
Accuracy: 82.18%
Accuracy of class 0 : 90.70%
Accuracy of class 1 : 97.40%
Accuracy of class 2 : 65.10%
Accuracy of class 3 : 84.70%
Accuracy of class 4 : 74.10%
Accuracy of class 5 : 78.70%
Accuracy of class 6 : 72.90%
Accuracy of class 7 : 90.20%
Accuracy of class 8 : 86.00%
Accuracy of class 9 : 82.00%

==================================================
2. Evaluating WHITE-BOX PGD Attack (eps=0.03137254901960784)
Accuracy: 0.00%
Accuracy of class 0 : 0.00%
Accuracy of class 1 : 0.00%
Accuracy of class 2 : 0.00%
Accuracy of class 3 : 0.00%
Accuracy of class 4 : 0.00%
Accuracy of class 5 : 0.00%
Accuracy of class 6 : 0.00%
Accuracy of class 7 : 0.00%
Accuracy of class 8 : 0.00%
Accuracy of class 9 : 0.00%

==================================================
3. Evaluating BLACK-BOX Transfer Attack (eps=0.03137254901960784)
Attack generated on SurrogateCNN, tested on ResNet-20
Accuracy: 49.86%
Accuracy of class 0 : 47.80%
Accuracy of class 1 : 77.70%
Accuracy of class 2 : 30.90%
Accuracy of class 3 : 63.40%
Accuracy of class 4 : 36.90%
Accuracy of class 5 : 47.80%
Accuracy of class 6 : 46.60%
Accuracy of class 7 : 43.90%
Accuracy of class 8 : 53.10%
Accuracy of class 9 : 50.50%
```

2. 代码注释

2.1 导入相应库及参数定义

导入要使用的库，并检查PyTorch以及CUDA是否可用。

- **定义对抗攻击参数**：`EPSILON` 限制扰动的最大范围，`PGD_ALPHA` 控制每次迭代的扰动步长，`PGD_ITERS` 决定攻击的迭代次数，数值越大攻击越强。
- 定义训练模型的保存位置。

In [ ]:
```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import os
import time # 时间库，用于计算训练耗时

# (1) 使用GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# (2) 对抗攻击参数
```

```
EPSILON = 8.0 / 255.0       # 最大扰动幅度
PGD_ALPHA = 2.0 / 255.0     # 单步扰动步长
PGD_ITERS = 10              # 攻击迭代次数

# (3) 训练参数
EPOCH = 30                  # ResNET-20模型的训练轮数
SURROGATE_MODEL_EPOCHS = 5  # 代理模型训练轮数
LEARNING_RATE = 0.001
BATCH_SIZE = 256

# (4) 目录和文件路径
DATA_DIR = './data'
TARGET_MODEL_PATH = os.path.join(DATA_DIR, 'resnet20.pth')
SURROGATE_MODEL_PATH = os.path.join(DATA_DIR, 'surrogate_model_cifar10.pth')
os.makedirs(DATA_DIR, exist_ok=True)
```

2.2 定义ResNet-20模型结构以及代理CNN的结构

- 定义 `ResNet-20` 模型架构;
- 定义代理模型：一个简单的CNN网络。

In [ ]:
```
class BasicBlock(nn.Module):
    # 残差定义：两个3X3卷积和残差连接
    expansion = 1
    def __init__(self, in_planes, planes, stride=1):
        super(BasicBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        # 残差连接，处理维度或步长不匹配的情况
        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion * planes, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion * planes)
            )
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out

class ResNet(nn.Module):
    # ResNet-20架构：含3个残差层
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_planes = 16
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(16)
        self.layer1 = self._make_layer(block, 16, num_blocks[0], stride=1) # Layer1:16通道，3个残差块
        self.layer2 = self._make_layer(block, 32, num_blocks[1], stride=2) # Layer2:32通道，步长2下采样
        self.layer3 = self._make_layer(block, 64, num_blocks[2], stride=2) #Layer3:64通道，步长2下采样
        self.fc = nn.Linear(64 * block.expansion, num_classes)
    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes * block.expansion
        return nn.Sequential(*layers)
    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = F.avg_pool2d(out, 8)  # 全局平均池化
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

def resnet20():
    return ResNet(BasicBlock, [3, 3, 3])
```

```
# 代理模型：简单的CNN
class SurrogateCNN(nn.Module):
    def __init__(self):
        super(SurrogateCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(64 * 8 * 8, 256)
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

2.3 加载数据集和预处理

- 攻击数据：不进行归一化，直接使用原始像素值（范围 [0,1]），便于扰动计算。
- 训练数据：通过随机裁剪和翻转增强数据，归一化使输入符合模型训练分布（均值为 0，标准差为 1）。

In [ ]:
```
cifar10_mean = (0.4914, 0.4822, 0.4465)
cifar10_std = (0.2470, 0.2435, 0.2616)

# (1) 用于攻击的转换，得到 [0,1] 范围的 Tensor
transform_attack = transforms.Compose([transforms.ToTensor()])

# (2) 用于模型训练和评估的转换，包含归一化
transform_train = transforms.Compose([
    transforms.RandomCrop(32, padding=4),    # 随机裁剪，增强泛化
    transforms.RandomHorizontalFlip(),       # 随机水平翻转
    transforms.ToTensor(),
    transforms.Normalize(cifar10_mean, cifar10_std)
])
transform_test = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(cifar10_mean, cifar10_std)
])

# (3) 数据集加载
# 攻击用的测试集 (不归一化)
test_dataset_attack = datasets.CIFAR10(root=DATA_DIR, train=False, download=True, transform=transform_at
test_loader_attack = DataLoader(test_dataset_attack, batch_size=1, shuffle=False)

# 训练和评估用的数据集 (归一化)
train_dataset_train = datasets.CIFAR10(root=DATA_DIR, train=True, download=True, transform=transform_tra
train_loader_train = DataLoader(train_dataset_train, batch_size=BATCH_SIZE, shuffle=True)
```

2.4 定义模型训练函数

- 使用交叉熵损失和Adam优化器；
- 训练结束后保存模型参数，便于后续加载评估。

In [ ]:
```
def train_model(model, train_loader, epochs, lr, model_path):
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)

    start_time = time.time()
    for epoch in range(epochs):
        model.train()
        running_loss = 0.0
        for i, (inputs, labels) in enumerate(train_loader):
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
        epoch_loss = running_loss / len(train_loader)
        print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}")

    end_time = time.time()
    print(f"Training Time Use: {end_time - start_time:.2f} seconds.")
    print(f"Saving model to {model_path}")
    torch.save(model.state_dict(), model_path)
```

2.5 PGD攻击函数的实现

- **迭代优化**：通过 `iters` 次迭代逐步调整扰动，使模型分类错误。
- **梯度符号法**：每次沿梯度符号方向更新扰动（ `alpha` 控制步长），最大化分类损失。
- **扰动限制**：通过 `clamp` 函数确保扰动幅度不超过 `eps` ，且像素值在合法范围。

In [ ]:
```
def pgd_attack(model, images, labels, eps, alpha, iters):
    images = images.clone().detach().to(device) # 复制原始图像（避免修改原值）
    labels = labels.clone().detach().to(device) # 保存原始图像用于扰动限制
    original_images = images.clone().detach()

    for i in range(iters):
        images.requires_grad = True
        # 因为模型输入要求，攻击时，需要对输入进行归一化
        images_normalized = transforms.Normalize(cifar10_mean, cifar10_std)(images)
        outputs = model(images_normalized)

        model.zero_grad()
        loss = F.cross_entropy(outputs, labels) # 分类损失
        loss.backward()

        # 沿使损失增加的方向，梯度符号法更新扰动
        adv_images = images + alpha * images.grad.sign()
        # 相对于原始图像，限制扰动在[-eps, eps]范围内
        eta = torch.clamp(adv_images - original_images, min=-eps, max=eps)
        # 确保对抗样本在合法像素范围[0,1]内
        images = torch.clamp(original_images + eta, min=0, max=1).detach()
    # 返回对抗样本
    return images
```

2.6 模型评估函数的实现

- **干净数据评估**：直接计算模型在原始测试集上的准确率。
- **白盒攻击评估**：使用目标模型生成对抗样本，评估模型对自身生成攻击的鲁棒性。
- **黑盒攻击评估**：使用代理模型生成对抗样本，评估对抗样本在目标模型上的迁移性。

In [ ]:
```
def evaluate_attack(model, loader, attack_fn=None, attack_kwargs=None, surrogate_model=None):
    correct = 0
    total = 0
    class_correct = list(0. for i in range(10))
    class_total = list(0. for i in range(10))

    model.eval()
    if surrogate_model:
        surrogate_model.eval()

    # loader 传入的是 test_loader_attack，其数据在 [0,1] 范围
    for images, labels in loader:
        images, labels = images.to(device), labels.to(device)
        # 生成对抗样本
        if attack_fn:
            # 黑盒攻击：使用代理模型生成对抗样本
            attack_model = surrogate_model if surrogate_model else model
            adv_images = attack_fn(attack_model, images, labels, **attack_kwargs)
            # 对抗样本需要归一化后才能输入模型
            input_images = transforms.Normalize(cifar10_mean, cifar10_std)(adv_images)
        else:
            # 干净样本，直接归一化
            input_images = transforms.Normalize(cifar10_mean, cifar10_std)(images)

        with torch.no_grad():
            outputs = model(input_images)
        _, predicted = torch.max(outputs.data, 1)# 获取预测类别
        total += labels.size(0)
        c = (predicted == labels).squeeze()
        correct += c.sum().item()

        # 计算每个类的准确率
        for i in range(labels.size(0)):
            label = labels[i]
            # 确保 c 是可迭代的
            current_c = c.item() if c.dim() == 0 else c[i].item()
            class_correct[label] += current_c
            class_total[label] += 1

    accuracy = 100 * correct / total
    print(f'Accuracy: {accuracy:.2f}%')

    for i in range(10):
        if class_total[i] > 0:
            print(f'Accuracy of class {i} : {100 * class_correct[i] / class_total[i]:.2f}%')

    return accuracy
```

2.7 加载模型与执行攻击

- 依次评估三种场景：干净数据、白盒攻击、黑盒攻击。
- 对比不同场景下的准确率，验证对抗攻击的有效性和迁移性。

In [ ]:
```
def main():
    # (1)准备目标模型(ResNet-20)
    print("="*50)
    print("Preparing Target Model (ResNet-20)...")
    target_model = resnet20().to(device)

    if os.path.exists(TARGET_MODEL_PATH):
        print(f"Loading previously trained target model from {TARGET_MODEL_PATH}")
        target_model.load_state_dict(torch.load(TARGET_MODEL_PATH, map_location=device))
    else:
        print("No pre-trained target model found. Starting training...")
        train_model(target_model, train_loader_train, EPOCH, LEARNING_RATE, TARGET_MODEL_PATH)

    # (2)准备代理模型
    print("\n" + "="*50)
    print("Preparing Surrogate Model (Custom CNN)...")
    surrogate_model = SurrogateCNN().to(device)

    if os.path.exists(SURROGATE_MODEL_PATH):
        print(f"Loading previously trained surrogate model from {SURROGATE_MODEL_PATH}")
        surrogate_model.load_state_dict(torch.load(SURROGATE_MODEL_PATH, map_location=device))
    else:
        print("No pre-trained surrogate model found. Starting training...")
        train_model(surrogate_model, train_loader_train, SURROGATE_MODEL_EPOCHS, LEARNING_RATE, SURROGAT


    # (3)评估
    print("\n" + "="*50)
    print("1. Evaluating on CLEAN data (Original Accuracy)")

    evaluate_attack(target_model, test_loader_attack)

    print("\n" + "="*50)
    print(f"2. Evaluating WHITE-BOX PGD Attack (eps={EPSILON})")
    pgd_params = {'eps': EPSILON, 'alpha': PGD_ALPHA, 'iters': PGD_ITERS}
    evaluate_attack(target_model, test_loader_attack, attack_fn=pgd_attack, attack_kwargs=pgd_params)

    print("\n" + "="*50)
    print(f"3. Evaluating BLACK-BOX Transfer Attack (eps={EPSILON})")
    print("Attack generated on SurrogateCNN, tested on ResNet-20")
    evaluate_attack(
        model=target_model,
        loader=test_loader_attack,
        attack_fn=pgd_attack,
        attack_kwargs=pgd_params,
        surrogate_model=surrogate_model
    )
```

```python
if __name__ == '__main__':
    main()
```