Blackbox_SurrogateNet_CIFAR10

May 26, 2025

1 黑盒攻击算法实现：Surrogate Network 对 CIFAR-10 数据集实现分类

1.0.1 Step1 导入依赖，设置运行模式，并加载 CIFAR-10 数据集

[1]:
```
# 导入 PyTorch 相关模块和数据加载工具
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# 导入 advertorch 中的 GenAttack 攻击方法
from advertorch.attacks.blackbox.gen_attack import LinfGenAttack

# tqdm 用于添加训练进度条
from tqdm import tqdm


# 设置计算设备，这里强制使用 CPU，避免与 GPU 环境冲突
device = torch.device("cpu")


# 定义图像预处理方式
transform = transforms.Compose([
    transforms.ToTensor()
])

# 下载并加载训练集和测试集
train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,␣
 ↪transform=transform)
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,␣
 ↪transform=transform)

# 构建 DataLoader，用于批量读取数据
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=False)
```

1.0.2 Step2 定义目标模型（Target Model）

[2]:
```
# 定义目标模型，用于被攻击
class TargetModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1), nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(64*16*16, 128), nn.ReLU(),
            nn.Linear(128, 10)
        )

    def forward(self, x):
        return F.softmax(self.conv(x), dim=1)
```

说明：• 定义目标模型（用于被攻击），使用了两层卷积层 + 一层池化层 + 两层全连接层，最后用 Softmax 函数输出每类概率。

1.0.3 Step3 定义代理模型（Surrogate Model）

[3]:
```
# 定义代理模型，用于模仿目标模型并发起黑盒攻击
class SurrogateModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1), nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Dropout(0.25),
            nn.Flatten(),
            nn.Linear(32*16*16, 64), nn.ReLU(),
            nn.Linear(64, 10)
        )

    def forward(self, x):
        return F.softmax(self.conv(x), dim=1)
```

说明：• 定义代理模型，目的是模仿目标模型的输出，从而执行黑盒攻击。

1.0.4 Step4 初始化模型并对目标模型进行训练

[4]:
```
# 实例化模型并放置在 CPU 上
target_model = TargetModel().to(device)
surrogate_model = SurrogateModel().to(device)
```

```python
# 定义训练函数
def train(model, loader, epochs=5, name="Model"):
    model.train()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)
    for epoch in range(epochs):
        loop = tqdm(loader, desc=f"[{name}] Epoch {epoch+1}/{epochs}",␣
 ↪leave=False)
        for x_batch, y_batch in loop:
            x_batch, y_batch = x_batch.to(device), y_batch.to(device)
            logits = model(x_batch)
            loss = F.cross_entropy(logits, y_batch)    # 交叉熵损失计算
            optimizer.zero_grad()    # 梯度置为 0
            loss.backward()    # 损失反向传播
            optimizer.step()  # 优化器迭代
            loop.set_postfix(loss=loss.item())  # tqdm 库 实时显示训练过程

# 开始训练目标模型
print("开始训练目标模型...")
train(target_model, train_loader, epochs=5, name="Target")
target_model.eval()
```

开始训练目标模型…

[4]:
```
TargetModel(
  (conv): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (5): Dropout(p=0.25, inplace=False)
    (6): Flatten(start_dim=1, end_dim=-1)
    (7): Linear(in_features=16384, out_features=128, bias=True)
    (8): ReLU()
    (9): Linear(in_features=128, out_features=10, bias=True)
  )
)
```

说明：• 对目标模型训练，进行交叉熵损失计算，设置训练轮次为 5 轮，训练结束后，用.eval() 切换到推理模式。

1.0.5 Step5 训练代理模型（知识蒸馏）

[5]:
```
# 使用目标模型的 soft label 训练代理模型
print("开始训练代理模型（模仿目标模型输出）...")
optimizer = torch.optim.Adam(surrogate_model.parameters(), lr=1e-3)
surrogate_model.train()

loop = tqdm(enumerate(train_loader), total=200, desc="Surrogate KD")
for i, (x_batch, _) in loop:
    if i >= 200:
        break
    x_batch = x_batch.to(device)
    with torch.no_grad():
        soft_labels = target_model(x_batch)
    preds = surrogate_model(x_batch)
    loss = F.mse_loss(preds, soft_labels)    # 均方误差损失计算
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    loop.set_postfix(loss=loss.item())

surrogate_model.eval()
```

开始训练代理模型（模仿目标模型输出）...

```
Surrogate KD: 100%|
                          | 200/200 [00:09<00:00, 20.80it/s,
loss=0.0538]
```

[5]:
```
SurrogateModel(
  (conv): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (3): Dropout(p=0.25, inplace=False)
    (4): Flatten(start_dim=1, end_dim=-1)
    (5): Linear(in_features=8192, out_features=64, bias=True)
    (6): ReLU()
    (7): Linear(in_features=64, out_features=10, bias=True)
  )
)
```

说明：• 通过知识蒸馏，使代理模型学习目标模型的输出分布（软标签）。• 设置训练前 200 个 batch，加快实验速度。

1.0.6 Step6 发起 GenAttack 黑盒攻击

[6]:
```
# 从测试集中获取一批图像
print("执行黑盒攻击（GenAttack）...")
x_test, y_test = next(iter(test_loader))
x_test = x_test.to(device)
y_test = y_test.to(device)

# 使用代理模型对目标模型执行黑盒攻击
attack = LinfGenAttack(
    predict=surrogate_model,
    eps=0.3,
    nb_samples=100,
    nb_iter=20,
    clip_min=0.0,
    clip_max=1.0,
    targeted=False
)

# 生成对抗样本
x_adv = attack.perturb(x_test, y_test)
```

执行黑盒攻击（GenAttack）…

说明：• LinfGenAttack 来源于 Github 上的 advertorch 库，属于黑盒攻击算法，不依赖于目标模型的梯度。• 此步骤通过代理模型对目标模型进行扰动攻击。

1.0.7 Step7 在目标模型上评估攻击效果

[7]:
```
# 在目标模型上评估攻击效果
print("评估攻击效果...")
with torch.no_grad():
    y_pred_clean = target_model(x_test).argmax(dim=1)
    y_pred_adv = target_model(x_adv).argmax(dim=1)

# 计算准确率和攻击成功率
acc_clean = (y_pred_clean == y_test).float().mean().item()
acc_adv = (y_pred_adv == y_test).float().mean().item()
attack_success_rate = (y_pred_clean != y_pred_adv).float().mean().item()

# 打印评估结果
print("原始样本准确率: {:.2%}".format(acc_clean))
print("对抗样本准确率: {:.2%}".format(acc_adv))
print("攻击成功率: {:.2%}".format(attack_success_rate))
```

评估攻击效果…
原始样本准确率: 65.62%
对抗样本准确率: 32.81%
攻击成功率: 56.25%

说明：• 原始样本准确率：目标模型对正常图像的分类准确率。• 对抗样本准确率：目标模型在被攻击后的样本上的准确率。• 攻击成功率：成功使模型预测改变的样本比例。

经过黑盒攻击实验，该Surrogate网络的CIFAR-10数据集识别准确率从65.62%下降至32.81%，证明实验结果正确，满足实现要求。