Adversarial Attack

胡若凡 3200102312

1. 背景

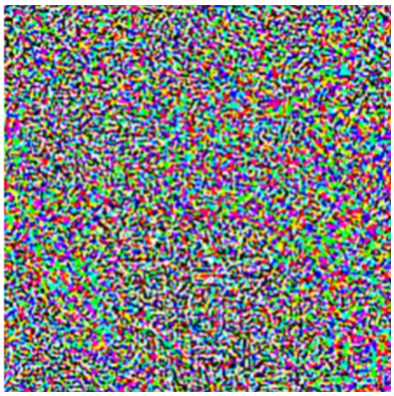
随着深度学习在各种应用领域的快速发展和显著成功，针对其的攻击也愈发严重。研究发现，由于深度神经网络（下称DNN）的黑盒性等原因，攻击者可以设计出针对样本的**微小扰动**，尽管它们对于人类可能是细微或者直观无影响的，但是却能导致输出结果的迥异性。

并且，对抗攻击基于以下的假设

- 攻击者只在测试和部署阶段攻击，并不在训练阶段攻击
- 攻击者攻击的模型是DNN模型
- 攻击者的目标是降低训练结果的精确度

例如，对下面的图例：

若对本张图添加对抗噪声：



那么将会把本张熊猫错误分类成长臂猿，并且该图片从视觉上改变不明显：

2. 论文阅读

阅读论文《Adversarial Examples Attacks and Defenses for Deep Learning》

《Towards Deep Learning Models Resistant to Adversarial Attacks》

2.1 领域名词

2.1.1 Threat Model

威胁模型可以进一步分解为四个方面：对抗性伪造、攻击者的知识、对抗特异性和攻击频率。下面对每个概念做一个记录

- **Adversarial Falsification**

  分为False positive和False negative两个种类，即伪造的结果

- **Adversary's Knowledge**

  分为black box和whte box两个种类，前者只知道输入与得到的结果，后者可以知道模型的具体设计细节

- **Adversarial Specificity**

  分为Targeted attacks和Nontargeted attacks，前者有明确的misguide导向，后者只要使输出的结果部署正确结果即可

  其中，Targeted attack如下：

$$\min_{x'} c(x, x') \quad \text{subject to:} \quad f(x') \neq y$$

  而NonTargeted attack如下：

$$\min_{x'} c(x, x') \quad \text{subject to:} \quad f(x') = y_T$$

- **Attack Frequency**

  分为One-time attacks和Iterative attacks，两者相比之下，相较于一次性攻击，迭代攻击通常可以产生更好的对抗性示例，但它需要更多的交互（更多的查询）与受害者分类器，并花费更多的计算时间来生成它们。对于一些计算密集型任务（例如，强化学习），一次性攻击可能是唯一可行的选择。

2.1.2 Perturbation

微小扰动是对抗性样本的一个基本前提。对抗性样本被设计为接近原始样本并且不可察觉的，这样才能达到对抗设计者的设计目的。下面对几个具体概念进行记录

- **Perturbation Scope**

  这里分为Individual attacks和Universe attacks。个体攻击为每个干净输入产生不同的扰动，通用攻击只为整个数据集创建一个通用扰动

- **Perturbation Limitation**

  这里分为Optimized perturbation和Constraint perturbation。优化扰动将扰动作为优化问题的目标。这些方法旨在最小化扰动，使得人类无法辨认扰动。 约束扰动将扰动设置为优化问题的约束条件

- **Perturbation Measurement**

  这里分为lp范数和PASS。举例而言，lp通过p-范数距离测量扰动的大小，计算在对抗性样例中改变的像素数，心理学感知对抗相似性分数（PASS）是最新的一种评判标准，其与人类感知一致，可以测量对抗性样例与原始样本之间的相似性。该度量标准是基于对称加权KL散度提出的。

2.1.3 Benchmark

数据集和受害模型的多样性，使得研究人员难以判断对抗样本的存在是由于数据集还是模型所致。下面记录常用的数据集与受害模型

- **Data Sets:**

  MNIST、CIFAR-10和ImageNet是最广泛使用的图像分类数据集，用于评估对抗性攻击。因为MNIST和CIFAR-10由于其简单性和小尺寸而易于攻击和防御，所以ImageNet是迄今为止最好的数据集来评估对抗性攻击。需要设计良好的数据集来评估对抗性攻击。

- **Victim Models**

  通常会攻击几个众所周知的深度学习模型，例如LeNet、VGG、AlexNet、GoogLeNet、CaffeNet和ResNet。

2.2 算法介绍

2.2.1 L-BFGS Attack

在L-BFGS Attack中，目标函数的输入是原始输入数据和错误分类的标签。错误分类的标签的产生是通过查询模型进行预测得到的，然后通过将其转化为one-hot编码来创建误分类的标签。

接下来，L-BFGS算法被用于最小化目标函数，并对c进行**linear-searching**生成对抗样本。在生成样本时，使用一些约束条件，例如添加噪声以确保对抗样本与原始输入类似。

$$\min_{x'} c\|\eta\| + J_\theta(x', l')$$
$$\text{s.t. } x' \in [0, 1].$$

2.2.2 FSGM

FGSM是梯度上升的单步（Fast）优化方法，优化方向沿着训练模型梯度下降的反方向。该算法中，假定深度网络容易收到对抗性扰动的主要原因是它们的线性性质，高维空间中的线性行为。如果我们在输入图像中加上计算得到的梯度方向，修改后的图像经过分类网络时的损失值就比修改前的图像经过分类网络时的损失值要大，模型预测对的概率就会变小。

2.2.3 PGD

在原始的 FGSM 中，攻击者只使用一个梯度来生成对抗性样本，这种攻击方式比较局限。PGD 在每个迭代步骤中都计算梯度，根据梯度进行多次更新，通过引入更多的扰动，从而产生更具有挑战性的对抗性样本。其算法公式如下：

$$(x,y)\sim\mathcal{D}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right]$$

2.2.4 JSMA

JSMA是一种基于雅可比矩阵的对抗样本攻击方法。它利用了图像中每个像素对模型输出的影响，从而创建对抗样本。

在这个方法中，雅可比矩阵是由神经网络的输出与输入相对应的梯度组成的。使用这个雅可比矩阵可以去导致每个像素对模型输出的影响。作者发现，通过每个样本仅修改4.02%的输入特征，实现了97% 的对抗性成功率。但是，由于其显着的计算成本，此方法运行非常缓慢。

2.2.5 DeepFool

DeepFool 的基本思想是将原始样本移动到最近的决策边界上，从而产生对抗性样本。它是一种迭代攻击方法，因此可以生成非常强大和具有挑战性的对抗性样本。

具体而言，DeepFool 方法会在决策边界上寻找一个最小扰动向量，该扰动向量将原始样本移到了另一类别的决策区域。这个扰动向量是通过计算网络梯度并沿着梯度方向进行更新生成的。然后，攻击者重复执行这个过程，直到达到预设的最大迭代次数或者满足某些收敛条件。

2.2.6 C&W's Attack

C&W's Attack 方法使用 L-BFGS（Limited-memory Broyden–Fletcher–Goldfarb–Shanno）算法，通过反复迭代来使得目标函数最小化，并满足上述约束。在每次迭代中，算法会根据当前的扰动向量对深度学习模型进行前向传播和反向传播，并利用梯度信息来更新扰动向量。

C&W's Attack 方法的优点在于通过优化来最小化扰动大小，从而在保证攻击成功率的同时尽可能减小扰动的幅度和可见性，使得攻击成功率和图像质量得到了有效的平衡

2.2.7 Zeroth-Order Optimization

Zeroth-Order Optimization，是一种零阶优化方法，用于生成对抗性样本。与其他对抗性攻击方法不同，ZOO 不需要直接访问或计算深度学习模型的梯度信息，而是基于模型输出的黑盒反馈信息来生成对抗样本。

Zeroth-Order Optimization 的基本思想是通过构造一个代理模型来对原始模型进行模拟，并根据模型输出的分类结果进行反馈，从而优化损失函数来生成对抗性样本。

2.2.8 Universal Perturbation

Universal Perturbation 是一种基于迭代算法的对抗性攻击方法，其主要目标是生成一组共性的扰动向量，使得这些扰动向量可以成功地攻击深度学习模型中的任意输入样本。下面我记录下对该方法流程的学习：

a. 随机选取一个未攻击成功的样本，并将其添加到所有已攻击成功的样本列表中。

b. 通过梯度下降或其他优化算法求解一个最小化扰动向量的问题，使得通用扰动向量可以成功地攻击所有已攻击成功的样本，并尽可能地减小扰动向量的幅度。

c. 如果通用扰动向量符合预设的停止条件，则跳出循环；

2.2.9 One-Pixel Attack

One-Pixel Attack 是一种基于遗传算法的对抗性攻击方法，其主要目标是在保证图像质量不变的情况下，通过修改图像中某个像素的 RGB 值来生成对抗性样本。与其他对抗性攻击方法不同，One-Pixel Attack 只修改了图像中的一个像素，因此攻击效果非常难以察觉。

3. 代码内容

3.1 攻击模型

这里我们针对CV来做一个经典的PGD算法，攻击的模型是MobileNet，数据集使用Mnist

```
class MobileNet(nn.Module):
    def __init__(self):
        super(MobileNet, self).__init__()
        net = models.mobilenet_v3_small(pretrained=False)
        #引入MobileNet-v3-small预训练模型并在其基础上进行微调。将pretrained参数设置为False表示不使用预训练权重,而是重新随机初始化权重
        self.trunk = nn.Sequential(*(list(net.children())[:-2]))
        #除了最后两层fc层之外的所有层,并以Sequential方式进行排列,即"backbone"
        self.avg_pool = nn.AdaptiveAvgPool2d(output_size=1)
        #定义一个自适应平均池化层,输出大小为1。该层的作用是将任意大小的输入张量转换为固定大小的输出张量。

        self.fc = nn.Sequential(
            nn.Linear(576, 10, bias=True),
            nn.LogSoftmax(dim=1)
        )
        #定义一个包含两个线性层和一个LogSoftmax层的序列模块。这是MobileNet模型的分类器

    def forward(self, x):
        #得到一个10维的概率向量。返回值为x
        x = self.trunk(x)
        x = self.avg_pool(x)
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x
```

3.2 PGD

PGD的代码的步骤为：

- 定义添加的扰动噪声delta，该delta即为最后输出时加在原图片上的扰动
- 开始迭代，迭代过程中，由于为 **白盒攻击** ，因此先用model的pred得到加了扰动后的图片与正确标签的CrossEntropyLoss，再根据loss进行调整。

  这里重要的是：由于python的loss函数自动是调整为向小的趋势，所以，我们希望与原分类差距越大。取个负号后，也就成了，**负数越大越小**，符合要求；而有目标攻击时，加上与target的loss，为正常的loss取法

```
def l_infinity_pgd(model, tensor, gt, epsilon=30./255, target=None, iter=100, show=True):
    '''
    model:白盒攻击的模型
    tensor:输入的图片,攻击的对象
    gt:输入图片tensor的正确分类
```

```
    epsilon:噪声扰动的范围,限制图片改变不至于过大
    target:确定是否为有目标or无目标攻击，不为None时为希望攻击的目标
    iter:迭代次数
    '''
    # 定义一个 delta 张量，与输入相同的形状，用于保存扰动
    delta = torch.zeros_like(tensor, requires_grad=True)
    # 定义一个 SGD 优化器，用于更新扰动
    opt = optim.SGD([delta], lr=10)

    for t in range(iter):
        # 对输入加上扰动，并对加扰动后的输入进行前向传播得到输出
        pred = model(norm(tensor + delta))
        # 如果没有指定目标类别，则采用原始标签；否则采用目标标签
        if target is None:
            loss = -nn.CrossEntropyLoss()(pred, torch.LongTensor([gt]))
        else:
            loss = loss = -nn.CrossEntropyLoss()(pred, torch.LongTensor([gt])) \
                + nn.CrossEntropyLoss()(pred, torch.LongTensor([target]))
        # 每 100 次迭代打印损失函数值
        if t % 100 == 0:
            print(t, loss.item())

        # 对扰动进行反向传播，计算梯度，并更新扰动
        opt.zero_grad()
        loss.backward()
        opt.step()
        # 将扰动裁剪到 [-epsilon, epsilon] 范围内
        delta.data.clamp_(-epsilon, epsilon)

    # 输出最终的预测结果和正确类别的概率
    print("True class probability:", nn.Softmax(dim=1)(pred))
    cnn_eval(norm(tensor+delta))

    # 如果 show 参数为 True，则显示不带扰动和加扰动后的图片
    if show:
        f,ax = plt.subplots(1,2, figsize=(10,5))
        ax[0].imshow((delta)[0].detach().numpy().transpose(1,2,0))
        ax[1].imshow((tensor + delta)[0].detach().numpy().transpose(1,2,0))

    # 返回加上扰动后的图片
    return tensor + delta
```

最终，利用该算法生成对抗攻击的样本，得到的正确率从98%下降到了74%

```
Test set: Average loss: 0.0696, Accuracy: 9788/10000 (98%)


Test set: Average loss: 0.7512, Accuracy: 1826/2453 (74%)
```

3.3 鲁棒性防御

对抗训练，数据增强、权重裁剪和Dropout正则化等方法，都可以提高模型的鲁棒性，下面给出了一段防御代码。并且， **FGSM**是仅仅对样本攻击一次的方法，这里由于是为了提高鲁棒性，不可以使用过多次迭代的图，并且幅度也需要较小，这是为了保证模型还是能学到正确的特征

```
def train(model, device, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()

        # 使用数据增强，如旋转、平移、缩放、翻转等
        data = data + torch.randn_like(data) * 0.1

        # 新增对抗样本，以增强模型的鲁棒性
        perturbed_data = fgsm_attack(model, data, target, epsilon=0.1)
        output = model(perturbed_data)

        # 权重裁剪
        parameters_to_prune = ((model.conv1, 'weight'), (model.conv2, 'weight'),
(model.fc1, 'weight'), (model.fc2, 'weight'))
        prune.global_unstructured(
            parameters_to_prune,
            pruning_method=prune.L1Unstructured,
            amount=0.1,
            )

        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
# 对抗攻击函数
def fgsm_attack(model, data, target, epsilon):
    data_raw = data.clone().detach().requires_grad_(True)
    output = model(data_raw)
    loss = F.nll_loss(output, target)
    model.zero_grad()
    loss.backward()
    perturbed_data = data_raw + epsilon * torch.sign(data_raw.grad)
    perturbed_data = torch.clamp(perturbed_data, 0, 1)
    return perturbed_data
```

4. 总结

深度学习领域之中，有众多模型研发，但是不可否认的是，很多模型在训练的时候都忽略了自身的鲁棒性，导致了模型有很大的安全漏洞，这就是对抗攻击与防御的目的。

这次的文档中，介绍了对抗攻击领域的相关知识，并且针对最经典的Mnist数据集，做了攻击与防御的实验。详细介绍的PGD是一种迭代式的攻击方式，其基本思想是在给定一个初始样本的情况下，使用多次迭代生成对抗样本。PGD攻击相比于其他攻击方式来说比较强大，可以有效地迫使模型产生误判，并且可以克服部分防御措施。

而为了防御，我们也可以使用对抗训练，数据增强、权重裁剪和Dropout正则化等方法，去提高模型的鲁棒性，但是其中的参数需要精细调整，并且运气也至关重要，这终究是牺牲了一部分正确率去以求争取较为稳定的模型结果。

在日后训练自己的深度学习模型时，都要更加关注模型的鲁棒性，以防被对抗攻击所影响。