对抗攻击与防御

学号 *519030910343* 姚瑾　*519030910348* 于本松

1. 任务概述

对抗攻击即通过添加不同的噪声或对图像的某些区域进行一定的改造生成对抗样本，此样本可以对网络模型进行攻击以达到混淆网络分类的目的。而训练鲁棒的网络模型可以有效地抵御攻击。

在任务一中，我们用攻击算法为指定的模型生成对抗样本，并尝试了随机化方法突破梯度掩蔽和改变损失函数。在任务二中，我们训练了一个鲁棒的神经网络模型，它可以有效抵御 PGD20 等算法的攻击。任务的训练与测试数据来源于 CIFAR10 数据集。

2. 任务一：白盒攻击竞赛

2.1. 主要攻击算法

2.1.1. FGSM（Fast Gradient Sign Method）算法

FGSM 算法通过梯度生成攻击噪声，也即在输入图像中加上计算得到的梯度方向，这样修改后的图像经过分类网络时的损失值就比修改前大，并通过 l_∞ 范数的约束确保前后图像的相似性，在本任务中距离约束 $||x_{adv}-x||_\infty < 8/255$：

$$X_{adv} = X + \epsilon * sign(\nabla_X L(X, y_{true})) \tag{1}$$

2.1.2. PGD（Projected Gradient Descent）算法

PGD 算法可以看作 FGSM 算法的翻版，只不过 PGD 算法有多次迭代，每次只走一小步并将扰动 clip 到规定范围，可以通过不断迭代算法得到攻击图像：

$$\begin{aligned} X_{adv}^0 &= X \\ X_{adv}^{N+1} &= Clip_{X,\epsilon}\Big\{X_{adv}^N + \alpha sign(\nabla_X L(X_{adv}^N, y_{true}))\Big\} \end{aligned} \tag{2}$$

2.1.3. DeepFool 算法

想要改变某点的分类结果，就是让其跨过分割平面，而最短移动距离就是垂直分割平面移动。DeepFool 算法即是在每次迭代时，使点以很小的移动距离不断逼近分割平面，下面的伪代码 Algorithm 1 可以较好解释这一过程：

Algorithm 1 DeepFool for multi-class case

Input: Image X,classifier f

Output: Perturbation Adv

Initialize $X_0 \leftarrow X$,$i \leftarrow 0$

while $\hat{k}(X_i) = \hat{k}(X_0)$ **do**

　　for $k \neq \hat{k}(X_0)$ **do**

　　$W_k^{'} \leftarrow \nabla f_k(X_i) - \nabla f_{\hat{k}(X_0)}(X_i)$

5: $f_k^{'} \leftarrow f_k(X_i) - f_{\hat{k}(X_0)}(X_i)$
6: **end for**
7: $\hat{l} \leftarrow argmin_{k \neq \hat{k}(X_0)} \frac{|f_k^{'}|}{||w_k^{'}||_2}$
8: $Adv_i \leftarrow \frac{|f_{\hat{l}}^{'}|}{||w_{\hat{l}}^{'}||_2^2} w_{\hat{l}}^{'}$
9: $X_{i+1} \leftarrow X_i^{'} + Adv_i$
10: $i \leftarrow i + 1$
11: **end while**
12: **return** $\hat{r} = \sum_i r_i$

2.2. 攻击结果

我们自己实现了 PGD、FGSM、DeepFool 攻击算法的代码，并调整接口使其可以通过运行 *attack_main.py* 文件运行。同时，通过调用对抗样本库 *advertorch* 中的 $SpatialTransformAttack$、$JacobianSaliencyMapAttack$、$LBFGSAttack$、$CarliniWagnerL2Attack$ 函数并调好参数接口进行攻击实验。

同时，针对提供的 PGD20 代码，我们采取了提前结束迭代的策略，在一些地方添加了随机化因子并加入了动量，在一定程度上提高了攻击算法的性能，此外还尝试了不同的损失函数，以及尝试了一些黑盒攻击的方法。

下面的表 1 显示了各个攻击算法的结果，其中 RunTime 是以 Model1 为例的攻击算法运行时间。这些结果均在实验室服务器上运行得到，服务器显卡是四张 GeForce RTX 3090 显卡：

Attack Method	Model1	Model2	Model3	Model4	Model5	Model6	BWD/FWD	RunTime
No Attack	0.9400	0.8300	0.8000	0.8500	0.8150	0.8800	-	-
PGD20	0.0004	0.5128	0.6462	0.5618	0.5482	0.6433	20/20	4min
EarlyEndPGD20	0.0004	0.5047	**0.1427**	0.5572	0.5396	0.6406	<20/<40	3.5min
SemiTargetedPGD20	0.0008	0.6847	0.1473	0.6508	0.6308	0.7035	<20/<40	3.5min
F-WEEPGD20	**0.0000**	**0.5012**	0.1921	**0.5529**	**0.5344**	**0.6381**	<20/<40	4min
DeepFool	0.0404	0.6406	0.1988	0.6902	0.6647	0.7457	200/20	50min
FGSM	0.2991	0.5663	0.5973	0.6087	0.6028	0.6794	1/1	0.38min
BlackBoxAttack	0.0004	0.8143	0.3149	0.8353	0.7989	0.8644	20/20	2min

表 1: 攻击实验数据

2.3. 攻击过程详述

2.3.1. EarlyEndPGD20: 提前结束迭代、随机化以及动量

在原始的 PGD 算法中，通过不断迭代得到攻击图像 X_{adv}，然而如果生成的图像已经可以被网络错误的分类，那么这些图像将没有必要继续被迭代，这样可以节省运算成本。

此外，我们发现提前结束的策略对基于梯度掩蔽的防御网络有比较好的作用，尽管它并没有完全克服梯度掩蔽这一问题，但却从另一个角度提供了问题的解决办法。总所周知，梯度掩蔽又称梯度混淆 (Obfuscated Gradients)，主要有三种方法：破碎梯度、随机梯度、爆炸及消失梯度，总的来讲，都是通过混淆原有的梯度，制造假梯度并且返回攻击者一个虚假的安全感来抵御攻击。基于梯度掩蔽防御的网络有多种

特征，比如一步攻击优于多步攻击，黑盒攻击优于白盒攻击等，这里我们所提出的提前结束迭代的策略正是基于一步攻击优于多步攻击这一原理。

通过对需要攻击的 6 个模型的结构的分析，我们发现 Model3 正是基于最后一个 cosinelinear 层实现的梯度掩蔽防御。因此，从理论上来讲一步攻击应当会优于多步攻击，我们应用了 PGD 算法的单步攻击版本 FGSM 算法，从表 1 可以看出 FGSM 的效果确实比 PGD20 效果要好，但提升相对不大，鲁棒准确率仍然接近 60%，因此我们提出了提前结束迭代的策略来进一步优化攻击结果。这一方法介于单步迭代与多步迭代，主要的创新点在于及时结束迭代，避免了因为梯度掩蔽所返回的错误梯度将已经得到的对抗样本再迭代回正确类别。主要方法为在每次迭代开始时检查一下上一步迭代得到的结果，将其中已经是对抗样本的图片取出来，这些图片将不再参与之后的迭代，剩下的图片则继续迭代更新，在最后一轮结束后将当前所有的样本都加入到结果中。将这一思想与随机化和动量结合以后，得到的 Model3 攻击结果鲁棒准确率仅为 0.1427，可以说有了相当大的改进。

更新对抗样本时我们加入了随机化因子和动量因素，随机化因子的一部分理论依据来源于攻击梯度掩蔽防御的黑盒攻击优于白盒攻击原理，我们在更新对抗样本时加入了比例为 0.008 的基于高斯分布生成的均值为 0 方差为 1 的随机矩阵。加入比例为 0.017 的动量因素主要是为了跳出局部最小值，加快收敛速度，防止在局部最小值或者鞍点处左右横跳的情况出现。

我们将这种结合了提前结束迭代策略、随机化以及动量的算法称为 $EarlyEndPGD20$，由上面的表格可以看出，$EarlyEndPGD20$ 算法与原始的 PGD20 算法相比在 Model3 有很大的性能提升，而在其他模型上也有不同程度的提升。整个算法的实现如 Algorithm 2 所示：

Algorithm 2 End Iteration Early for PGD

Input: Image x
Output: Adversarial example $result_k$

Initialize $x_0 \leftarrow x$
for $i = 0, \ldots\ldots, k-1$ **do**
  Compute $f(x)$ and Loss $L(f(x), y)$
  Backward network:$\nabla_x L(x)$
  for $image$ in x_i **do**
    if $Label(image)$ is true **then**
      $result_i \leftarrow image$
    else
      $notsucceed\_attack_i \leftarrow image$
    end if
  end for
  $notsucceed\_attack_{i+1}$=$notsucceed\_attack_i$+α sign$(\nabla_x L(x))$
  $notsucceed\_attack_{i+1}$=$notsucceed\_attack_{i+1}$+$\beta_1 * randomization\_factor$+$\beta_2 * grad\_tot$
  $notsucceed\_attack_{i+1}$=$Clip_{notsucceed\_attack,\epsilon}(notsucceed\_attack_{i+1})$
end for
$result_k$=$result_k$+$notsucceed\_attack_k$

2.3.2. SemiTargetedPGD20：调整损失函数以实现半定向攻击

$SemiTargetedPGD20$ 算法主要是我们基于 $EarlyEndPGD20$ 算法所得出的另一个攻击算法，它通过更改损失函数，将攻击从原来的无目标攻击变为攻击到当前错误概率最大的类别，具体方法如下：由于我们每一轮迭代的最开始时会检查当前样本是否为对抗样本，并且将已经是对抗样本的样本取出，因此剩下的样本均为第一预测类别正确的样本，那么我们采取该样本的第二预测类别作为我们的目标攻击到的类

别，在算法中主要实现了一个 second_max(y_hat) 函数，该函数可以返回矩阵维度 1 中第二大元素的下标，之后取

$$loss\_c = -F.cross\_entropy(y\_hat, second\_max(y\_hat)) \tag{3}$$

，其中 y_hat 为模型对当前 X_{adv} 的预测结果，其他步骤与 $EarlyEndPGD20$ 算法相同。

通过比较该算法与 $EarlyEndPGD20$ 算法和原始 PGD20 的结果，可以发现该算法只有在 Model3 上攻击的结果较好，与 $EarlyEndPGD20$ 算法结果相近，而其他几个模型的结果均不如 $EarlyEndPGD20$ 算法和原始 PGD20，我们认为其在 Model3 上的表现较好主要原因在于该算法中应用的提前结束迭代策略，这也验证了 $EarlyEndPGD20$ 算法中提出的提前结束迭代策略针对梯度掩蔽防御的有效性。尽管 $SemiTargetedPGD20$ 算法在其他模型上表现稍差，但我们认为该算法仍然有较大的提升空间，之后的工作可以从损失函数的进一步优化，比如为了防止步长过大导致对抗样本在靠近 2 个错误类之间摇摆，可以令迭代策略为每 2-3 轮迭代更新一次目标类，还可以从样本更新的方面进行改进，比如引入自适应步长等，这些工作有待于后续研究。

2.3.3. BlackBoxAttack：一个模型生成的对抗样本攻击另一个模型

BlackBoxAttack 算法是基于原始 PGD20 算法得到的黑盒攻击算法，它主要的策略是对于一个黑盒模型，我们用相同的数据训练一个类似的神经网络，然后用 PGD20 算法生成自己训练的白盒网络对抗样本，然后用该样本去攻击黑盒网络，有相应的研究已经表明该方法是可行的。由于我们要攻击的 6 个模型中模型 1 最容易攻击并且将近 100% 正确，我们便将该模型作为白盒模型，生成对抗样本的算法使用原始的 PGD20 算法，因为 Model1 没有使用梯度掩蔽，生成对抗样本比较容易，如果使用 $EarlyEndPGD20$ 算法，则生成的对抗样本在数据分布边界，如果黑盒模型鲁棒性比较好的话则攻击成功率比较低。

从结果可以看出，*BlackBoxAttack* 算法起到的作用非常小，除去 Model1 以外的 5 个模型中，只有模型 3 攻击的效果比较显著，达到了 31%，其他几个均是在无攻击情况下提升了一两点，这也进一步验证了模型 3 是基于梯度掩蔽实现的，并且验证了针对梯度掩蔽黑盒攻击优于白盒攻击的结论。其他几个模型效果比较差的原因，可能是因为我们构造的网络和黑盒网络结构差异比较大，如果采用相同的网络结构训练的话，结果或许会有提升。

2.3.4. F-WEEPGD20：采用 *Frank-Wolfe* 迭代方法

$F-WEEPGD20$ 算法是基于与 $EarlyEndPGD20$ 算法的改进版本，$F-WEEPGD20$ 算法改进了每次迭代生成新样本的方式，它采用了 Frank-Wolfe 约束优化方法，Frank-Wolfe 方法的基本思想是：每次迭代中使用一阶泰勒展开式将目标函数线性化，通过解线性规划得到可行方向，进而沿此方向在可行域内作一维搜索。这一方法主要参考了时若曦同学的思路，同时也查找了一些资料。[1]

从结果上来看 $F-WEEPGD20$ 算法和之前的结果相比有了一定的改进，除了 Model3 的结果没有 $EarlyEndPGD20$ 算法好，其他几个都比 $EarlyEndPGD20$ 算法要提升一点，这让我明白了优化方法的重要性，合适的优化方法往往能给深度学习模型带来性能上的提升。

2.4. 攻击实验

2.4.1. 探究 PGD 迭代次数与攻击效果的关系

我们使用鲁棒性较差的 Model3 作为被攻击的算法，更改 EarlyEndPGD 攻击算法的迭代次数，得到图 1 中所示的结果。可以看出在计算资源允许的条件下，攻击算法的迭代次数越多，攻击效果越好，但在迭代次数已经很大时增大迭代次数带来的收益不够明显。

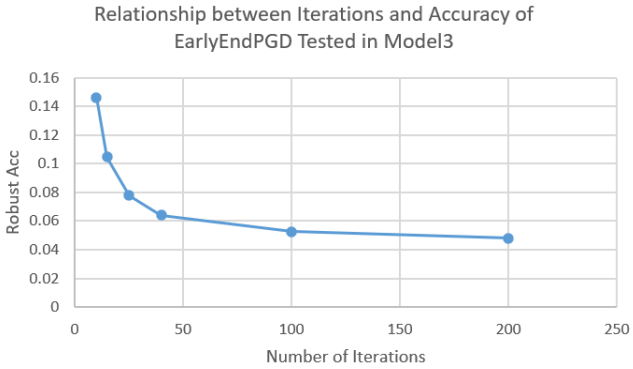

图 1: *PGD* 迭代次数与攻击效果的关系

2.4.2. 探究 PGD 算法添加动量和随机因子的作用效果

我们通过一个动量系数乘 torch.sign(grad_tot) 生成动量值，并将其加到更新的模型上以更好的跳出局部极小。图 2 左侧是使用原始 PDG 算法添加不同动量系数的攻击结果，可以看出存在一个 0.015 到 0.02 之间的动量系数使得攻击效果最好。

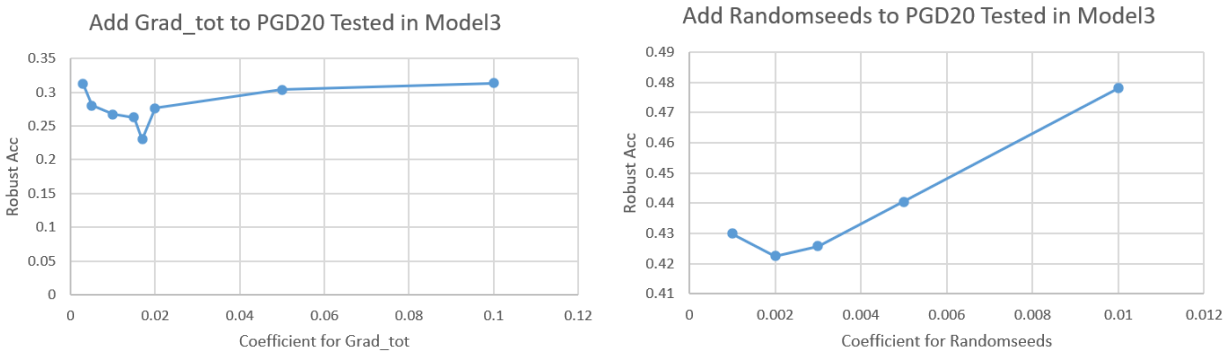

图 2: 添加动量与随机因子的实验

我们通过一个随机因子系数乘 torch.randn(x_adv.shape) 生成随机数，虽然理论上这有助于突破梯度掩蔽，但是实验发现随机因子的作用不大，甚至有时起到反作用。由于计算资源的限制，我们只在 Model3 上进行了测试，如图 2 右图所示。综合以上探索，我们认为在这个算法中，动量起到了比较大的作用，而随机因子则作用相对较小。

我们还发现了一个有趣的现象，在我们用 1 张 1080 ti 显卡运行 *SemiTargetedPGD*20 算法攻击 Model3 时，意外地得到了只有 0.068 的鲁棒准确率，但在后来用其他多个设备运行相同的代码时，得到的结果只有 0.14，出于科学严谨的态度，我们以 0.14 为准，我们猜想可能是因为 1080 ti 显卡有独特的优化机制。

3. 任务二：训练一个鲁棒的神经网络模型

3.1. 主要防御方法

3.1.1. 对抗训练（*Adversarial Training*）

对抗训练是增强神经网络鲁棒性的重要方式，在对抗训练中，样本会被我们的攻击算法混合对应的扰动，然后使神经网络适应这种改变，从而增加分类模型的鲁棒性，从 *normal_train.py* 和 *pgd_train.py* 的训练和测试准确率对比来看，这个方法较为有效，方法的实现如 Algorithm3 所示：

Algorithm 3 PGD Adversarial Training

Input: Step size η_1 and η_2,Random start radius α,Batch size m,Number of iterations K,Network architecture parametrized by θ

Output: Robust network f_θ

Initialize f_θ
repeat
  Read mini-batch B=$x_1,\cdots\cdots,x_m$ from training set
  Random re-start for PGD:
  for 1,$\cdots\cdots$,K **do**
    $x^{'} \leftarrow \pi_{B(x,\epsilon)}(\eta_1 * sign(\nabla_{x'} L(f_\theta(x), y)) + x')$
  end for
  $\theta \leftarrow \theta - \eta_2 \nabla_\theta L(f_\theta(x^{'}), y)$
until Training converged

3.1.2. 正则化

在优化目标函数时，除了正常的损失函数外，为了防止过拟合，通常会加入一些正则项，可以在对抗训练的基础上引入更强的正则化。

3.2. 防御结果

下表为我们所采取的 3 种防御训练方式并用两种攻击算法攻击所得出的最终结果，由于算力限制，关于正则化的探索并未完整训练模型，故这里不给出关于正则化的训练结果。使用的服务器有两张 GeForce RTX 3090，但可能有一部分算力被其他人的程序占用，因此时间的参考价值有限。

Defence Network	Attack	ACC	Robust_ACC	BWD/FWD	RunTime
Baseline_Resnet18	PGD	0.8309	0.5171	5×10^7	4h
Changed_Baseline_Resnet18	PGD	0.8400	0.5001	5×10^7	4h
	F-WEEPGD20	0.8400	0.4870	5×10^7	4h
Ours_ObscureGradient_Resnet18	PGD	0.8040	0.4946	5×10^7	4h
	F-WEEPGD20	0.8040	0.4831	5×10^7	4h
Trades-AWP_WideResnet28	PGD	0.8640	0.5832	$10^7/1.3 \times 10^8$	35h
	F-WEEPGD20	0.8640	0.5753	$10^7/1.3 \times 10^8$	35h

表 2: 防御实验数据

3.3. 防御过程详述

3.3.1. 正则化

我们尝试了引入更强的正则项，但结果显示增大正则项的系数为 5×10^{-3} 和 10^{-3} 会使最终对抗训练的结果下降。此外对抗训练所用时间比较久，我们只能依据前几个 epoch 判断最终的可能效果，因此放弃了这方面的继续探索。

3.3.2. 更改网络参数

我们尝试增大 PGD 攻击的距离约束，将 8/255 改为 16/255，这样生成的对抗样本距离原图像更远，将这样生成的对抗样本作为训练数据，理论上训练出来的模型对噪声的鲁棒性应该更强，尽管它可能也避免不了对抗训练的过拟合缺点。

我们采用了 Resnet18 作为训练网络，训练 100 个 epoch 后发现最好的鲁棒准确率为 0.5001，比 baseline 的距离约束 8/255 结果 0.5171 稍差一点，我们分析，这可能是因为引入的噪声距离太大，由此训练出来的网络针对训练的对抗样本过拟合太严重，导致数据分布与原数据分布出现偏差，导致最终的鲁棒性下降。从理论上来看，改变距离约束从一定程度上能优化最终模型的鲁棒性，因此我们认为如果算力足够，不断调试距离约束，最终可以找出较好的距离约束从而改善模型鲁棒性。

3.3.3. 梯度掩蔽

通过对攻击任务的探索，我们发现 PGD 算法针对梯度掩蔽的效果比较差，因此我们考虑用梯度掩蔽来防御攻击。我们采用了 Resnet18 作为训练网络，但在最后的线性层参考了 Model3 的 coslinear 层，试图构造一个类似的梯度掩蔽网络，训练时也考虑了增大距离约束为 16/255，最终训练得到的结果为 0.4946，同样不如 baseline 的结果，这可能是由于是不同模型训练的结果会有一定的差异，因此我们训练的结果不如 Model3 那么高。

3.3.4. Trades,Adversarial Weight Perturbation

通过对模型 4、5、6 的研究，我们决定采用和这几个模型类似的方法来训练防御模型。所采用的方法是 Trades-AWP 的方法，它采用了 trades 作为损失函数，以及 adversarial weight perturbation 的训练迭代机制，训练的网络为 Trades-WideResnet28，代码实现主要参考了 <Adversarial Weight Perturbation Helps Robust Generalization> 的官方给出的代码。[2]

在训练 100 个 epoch 的情况下，Robust_acc 在第 91 个 epoch 时达到最大，为 58.32%，限于计算资源，没有继续训练下去，鉴于观察训练的日志，在训练时 loss 仍在下降，训练准确率仍有提升的趋势，test_robus_acc 处于波动的状态，并没有发生下降，因此我们猜想该方法的 Robust_acc 或许还能提高，当然一个更值得思考的问题是如何才能改进该方法加快收敛速度，这可以作为之后的研究内容。

4. References

[1] Wikipedia, “Frank–wolfe algorithm,” https://en.wikipedia.org/wiki/Frank%E2%80%93Wolfe_algorithm.

[2] D. Wu, S.-T. Xia, and Y. Wang, “Adversarial weight perturbation helps robust generalization,” in *NeurIPS*, 2020.